

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

ЛАБОРАТОРНА РОБОТА №1

з дисципліни «Паралельні та розподілені обчислення

Паралельне програмування-2»

Тема: «Ада. Семафори»

Виконав:

студент 3-го курсу

групи ІП-42

з номер заліковки 4206

Дзюба Влад

Київ – 2016

**Завдання**

**Мета роботи:** розробка програми для ПКС зі СП

**Мова програмування:** Ада

**Засоби організації взаємодії процесів:** семафори мови Ада з пакета

Ada.Synchronous\_Task\_Control.

**Варіант:**

Пристрої

вводу-виводу

ФП

1

2

**Математичний паралельний алгоритм:**

- будь-який алгоритм сортування (в данній роботі це — алгоритм бульбашки).

- алгоритм сортування злиттям.

СР: e, T, MK.

**Алгоритм для кожного процесу:**

|  |  |
| --- | --- |
| 1. Ввести e, T, Z, MK, MR.  2. Розділити Z та MR на частини та .  3. Сигнал з до .  4. Обрахувати .  5. Чекати сигнал з до.  6. Обрахувати .  7. Вивести . | 1. Чекати сигнал з до.  2. Обрахувати .  3. Сигнал з до . |

**Структурна схема взаємодії процесів**

**Лістінг програми:**

Sem1

Sem2

СР

Skd

**vector.ads:**

--------------------------------------------------------------------------------

-- vector

-- Purpose:

-- Use arrays with dynamic index

--------------------------------------------------------------------------------

generic

type Index is range <>;

package vector is

type Vector is array (Index) of Integer;

function mult(k: Integer; vec: Vector) return Vector;

------------------------------------------------------------------------------

-- oneVector

-- Purpose:

-- Returns vector with all elements equal one

------------------------------------------------------------------------------

function oneVector return Vector;

procedure sort(vec: in out Vector);

procedure print(vec: Vector);

end vector;

**vector.adb:**

with Ada.Text\_IO; use Ada.Text\_IO;

package body vector is

function oneVector return Vector is

res: Vector;

begin

for i in Index loop

res(i) := 1;

end loop;

return res;

end;

function mult(k: Integer; vec: Vector) return Vector is

res: Vector;

begin

for i in res'Range loop

res(i) := k \* vec(i);

end loop;

return res;

end;

procedure sort(vec: in out Vector) is

c: Integer;

begin

for i in vec'First+1..vec'Last loop

for j in vec'First..vec'Last - (i - vec'First) loop

if vec(j) > vec(j + 1) then

c := vec(j);

vec(j) := vec(j + 1);

vec(j + 1) := c;

end if;

end loop;

end loop;

end;

procedure print(vec: Vector) is

begin

for i in vec'Range loop

Put(Integer'Image(vec(i)));

Put(" ");

end loop;

New\_Line;

end;

end vector;

**matrix.ads:**

--------------------------------------------------------------------------------

-- matrix

-- Purpose:

-- Using matrixes as array with dynamic index of vectors.

--------------------------------------------------------------------------------

with vector;

generic

type Index is range <>;

with package vector\_i is new vector(<>);

package matrix is

use vector\_i;

type Matrix is array (Index) of vector\_i.Vector;

function sum(mat1: Matrix; mat2: Matrix) return Matrix;

------------------------------------------------------------------------------

-- oneMatrix

-- Purpose:

-- Creating matrix with each element equals one.

------------------------------------------------------------------------------

function oneMatrix return Matrix;

------------------------------------------------------------------------------

-- fromVector

-- Purpose:

-- Creating one-height matrix from Vector

------------------------------------------------------------------------------

function fromVector(vec: vector\_i.Vector) return Matrix;

------------------------------------------------------------------------------

-- toVector

-- Purpose:

-- Creating vector from matrix. Returns first row of matrix.

------------------------------------------------------------------------------

function toVector(mat: Matrix) return vector\_i.Vector;

procedure print(mat: Matrix);

end matrix;

**matrix.adb:**

with Ada.Text\_IO; use Ada.Text\_IO;

with Ada.Integer\_Text\_IO; use Ada.Integer\_Text\_IO;

package body matrix is

function oneMatrix return Matrix is

res: Matrix;

begin

for i in Index loop

res(i) := oneVector;

end loop;

return res;

end;

function fromVector(vec: vector\_i.Vector) return Matrix is

res: Matrix;

begin

for i in res'Range loop

for j in res(i)'Range loop

res(i)(j) := vec(j);

end loop;

end loop;

return res;

end;

function sum(mat1: Matrix; mat2: Matrix) return Matrix is

res: Matrix;

begin

for i in res'Range loop

for j in res(i)'Range loop

res(i)(j) := mat1(i)(j) + mat2(i)(j);

end loop;

end loop;

return res;

end;

function toVector(mat: Matrix) return vector\_i.Vector is

res: vector\_i.Vector;

begin

for i in res'Range loop

res(i) := mat(mat'First)(i);

end loop;

return res;

end;

procedure print(mat: Matrix) is

begin

for i in mat'Range loop

vector\_i.print(mat(i));

end loop;

end;

end matrix;

**lib.ads:**

--------------------------------------------------------------------------------

-- lib

-- Purpose:

-- Using functions linked to work with vector or matrixes with different

-- indexes.

--------------------------------------------------------------------------------

with Ada.Synchronous\_Task\_Control; use Ada.Synchronous\_Task\_Control;

with vector;

with matrix;

package lib is

------------------------------------------------------------------------------

-- sliceVec

-- Purpose:

-- Get i-th part of vector.

------------------------------------------------------------------------------

generic

type IndexI is range <>;

type IndexO is range <>;

with function convert(partI: Integer; ind: IndexO) return IndexI;

with package vector\_i is new vector(IndexI);

with package vector\_o is new vector(IndexO);

function sliceVec(partI: Integer; vec: vector\_i.Vector) return vector\_o.Vector;

-----------------------------------------------------------------------------------------

-- sliceMat

-- Purpose:

-- Get i-th part of matrix. Matrix slices by columns (height stays the same).

-----------------------------------------------------------------------------------------

generic

type Index is range <>;

type IndexI is range <>;

type IndexO is range <>;

with function convert(partI: Integer; ind: IndexO) return IndexI;

with package vector\_i\_i is new vector(IndexI);

with package vector\_i\_o is new vector(IndexO);

with package matrix\_i is new matrix(Index, vector\_i\_i);

with package matrix\_o is new matrix(Index, vector\_i\_o);

function sliceMat(partI: Integer; mat: matrix\_i.Matrix) return matrix\_o.Matrix;

generic

type RowIndex1 is range <>;

type GenIndex is range <>;

type ColIndex2 is range <>;

with package vector\_i1 is new vector(GenIndex);

with package vector\_i2 is new vector(ColIndex2);

with package matrix1 is new matrix(RowIndex1, vector\_i1);

with package matrix2 is new matrix(GenIndex, vector\_i2);

with package matrix\_o is new matrix(RowIndex1, vector\_i2);

function mult(mat1: matrix1.Matrix; mat2: matrix2.Matrix) return matrix\_o.Matrix;

-----------------------------------------------------------------------------------------

-- merge\_p

-- Purpose:

-- Merge matrix's sorted vectors into one big sorted vector

-----------------------------------------------------------------------------------------

generic

with package matrix\_i is new matrix(<>);

with function newSize(s1: matrix\_i.Index; s2: matrix\_i.vector\_i.Index) return Integer;

package merge\_p is

use matrix\_i;

use type Index;

use type vector\_i.Index;

subtype IndexO is Integer range 0..newSize(Index'Last-Index'First, vector\_i.Index'Last - vector\_i.Index'First)-1;

package vector\_o is new vector(IndexO);

function mergeVectors(vec: matrix\_i.Matrix) return vector\_o.Vector;

end merge\_p;

end lib;

**lib.adb:**

with Ada.Synchronous\_Task\_Control; use Ada.Synchronous\_Task\_Control;

with Ada.Text\_IO; use Ada.Text\_IO;

package body lib is

function sliceVec(partI: Integer; vec: vector\_i.Vector) return vector\_o.Vector is

res: vector\_o.Vector;

begin

for i in res'Range loop

res(i) := vec(convert(partI, i));

end loop;

return res;

end;

function sliceMat(partI: Integer; mat: matrix\_i.Matrix) return matrix\_o.Matrix is

res: matrix\_o.Matrix;

begin

for i in res'Range loop

for j in res(i)'Range loop

res(i)(j) := mat(i)(convert(partI, j));

end loop;

end loop;

return res;

end;

function mult(mat1: matrix1.Matrix; mat2: matrix2.Matrix) return matrix\_o.Matrix is

res: matrix\_o.Matrix;

begin

for i in res'Range loop

for j in res(i)'Range loop

res(i)(j) := 0;

for k in mat2'Range loop

res(i)(j) := res(i)(j) + mat1(i)(k) \* mat2(k)(j);

end loop;

end loop;

end loop;

return res;

end;

package body merge\_p is

function mergeVectors(vec: matrix\_i.Matrix) return vector\_o.Vector is

vec\_i: array (Index) of vector\_i.Index;

vec\_b: array (Index) of Boolean;

res: vector\_o.Vector;

res\_i: Integer := 0;

min\_val: Integer;

min\_j: Index := Index'First;

begin

for i in vec\_i'Range loop

vec\_i(i) := vec(i)'First;

vec\_b(i) := true;

end loop;

for i in res'Range loop

min\_j := vec'First;

min\_val := 32000;

for j in vec'Range loop

if vec\_b(j) and vec(j)(vec\_i(j)) < min\_val then

min\_j := j;

min\_val := vec(j)(vec\_i(j));

end if;

end loop;

res(res\_i) := vec(min\_j)(vec\_i(min\_j));

if vec\_i(min\_j) = vector\_i.Index'Last then

vec\_b(min\_j) := false;

else

vec\_i(min\_j) := vec\_i(min\_j) + 1;

end if;

res\_i := res\_i + 1;

end loop;

return res;

end;

end merge\_p;

end lib;

**lib.adb:**

with Ada.Synchronous\_Task\_Control; use Ada.Synchronous\_Task\_Control;

with Ada.Text\_IO; use Ada.Text\_IO;

package body lib is

function sliceVec(partI: Integer; vec: vector\_i.Vector) return vector\_o.Vector is

res: vector\_o.Vector;

begin

for i in res'Range loop

res(i) := vec(convert(partI, i));

end loop;

return res;

end;

function sliceMat(partI: Integer; mat: matrix\_i.Matrix) return matrix\_o.Matrix is

res: matrix\_o.Matrix;

begin

for i in res'Range loop

for j in res(i)'Range loop

res(i)(j) := mat(i)(convert(partI, j));

end loop;

end loop;

return res;

end;

function mult(mat1: matrix1.Matrix; mat2: matrix2.Matrix) return matrix\_o.Matrix is

res: matrix\_o.Matrix;

begin

for i in res'Range loop

for j in res(i)'Range loop

res(i)(j) := 0;

for k in mat2'Range loop

res(i)(j) := res(i)(j) + mat1(i)(k) \* mat2(k)(j);

end loop;

end loop;

end loop;

return res;

end;

package body merge\_p is

function mergeVectors(vec: matrix\_i.Matrix) return vector\_o.Vector is

vec\_i: array (Index) of vector\_i.Index;

vec\_b: array (Index) of Boolean;

res: vector\_o.Vector;

res\_i: Integer := 0;

min\_val: Integer;

min\_j: Index := Index'First;

begin

for i in vec\_i'Range loop

vec\_i(i) := vec(i)'First;

vec\_b(i) := true;

end loop;

for i in res'Range loop

min\_j := vec'First;

min\_val := 32000;

for j in vec'Range loop

if vec\_b(j) and vec(j)(vec\_i(j)) < min\_val then

min\_j := j;

min\_val := vec(j)(vec\_i(j));

end if;

end loop;

res(res\_i) := vec(min\_j)(vec\_i(min\_j));

if vec\_i(min\_j) = vector\_i.Index'Last then

vec\_b(min\_j) := false;

else

vec\_i(min\_j) := vec\_i(min\_j) + 1;

end if;

res\_i := res\_i + 1;

end loop;

return res;

end;

end merge\_p;

end lib;

**calc\_p.ads:**

--------------------------------------------------------------------------------

-- calc\_p

-- Purpose:

-- Calculate formula of lab task.

--------------------------------------------------------------------------------

with vector;

with matrix;

generic

type IndexZ is range <>;

with package vectorZ is new vector(IndexZ);

type IndexT is range <>;

with package vectorT is new vector(IndexT);

type IndexKR is range <>;

with package vectorK is new vector(IndexKR);

with package matrixK is new matrix(IndexT, vectorK);

with package matrixR is new matrix(IndexKR, vectorZ);

package calc\_p is

procedure calc(e: Integer; Z: vectorZ.Vector; T: vectorT.Vector; K: matrixK.Matrix; R: matrixR.Matrix; Ah: out vectorZ.Vector);

end calc\_p;

**tasks.ads:**

--------------------------------------------------------------------------------

-- tasks

-- Purpose:

-- Using task type for calculating vector part

--------------------------------------------------------------------------------

with Ada.Synchronous\_Task\_Control; use Ada.Synchronous\_Task\_Control;

with vector;

with matrix;

with calc\_p;

generic

with package calc\_p\_i is new calc\_p(<>);

use calc\_p\_i;

package tasks is

task type calc\_part is

entry start(sem: in out Suspension\_Object; e: Integer; Z: vectorZ.Vector; T: vectorT.Vector; K: matrixK.Matrix; R: matrixR.Matrix; Ah: out vectorZ.Vector);

end calc\_part;

end tasks;

**tasks.adb:**

with Ada.Text\_IO; use Ada.Text\_IO;

package body tasks is

task body calc\_part is

begin

accept start(sem: in out Suspension\_Object; e: Integer; Z: vectorZ.Vector; T: vectorT.Vector; K: matrixK.Matrix; R: matrixR.Matrix; Ah: out vectorZ.Vector) do

calc\_p\_i.calc(e, Z, T, K, R, Ah);

Set\_True(sem);

end start;

end calc\_part;

end tasks;

**main.adb:**

--------------------------------------------------------------------------------

-- main

-- Author:

-- Dzyuba Vlad, IP-42

-- Purpose:

-- Parallel calculating of operations with numbers, vectors and matricies.

--------------------------------------------------------------------------------

with Ada.Text\_IO; use Ada.Text\_IO;

with Ada.Synchronous\_Task\_Control; use Ada.Synchronous\_Task\_Control;

with lib; use lib;

with tasks;

with calc\_p;

with matrix;

with vector;

procedure main is

-- Output vector size

N: constant Integer := 6;

-- Number of processors

P: constant Integer := 3;

-- Size of one vector part

pSize: constant Integer := N / P;

-- MK row width and MR column height

N2: Integer := 4;

-- Vector T width

N3: Integer := 5;

subtype IndexZ is Integer range 0..N-1;

subtype IndexT is Integer range 0..N3-1;

subtype IndexKR is Integer range 0..N2-1;

package vectorZ is new vector(IndexZ);

package vectorT is new vector(IndexT);

package vectorK is new vector(IndexKR);

package matrixK is new matrix(IndexT, VectorK);

package matrixR is new matrix(IndexKR, vectorZ);

-- Vector and matrix types for part formula

subtype IndexPart is IndexZ range 0..pSize-1;

package vectorZh is new vector(IndexPart);

subtype VecIndex is Integer range 0..0;

package matrixZh is new matrix(VecIndex, VectorZh);

package matrixRh is new matrix(IndexKR, VectorZh);

-- Input number, vectors and matricies

e: Integer := 1;

Z: vectorZ.Vector := vectorZ.oneVector;

T: vectorT.Vector := vectorT.oneVector;

MK: matrixK.Matrix := matrixK.oneMatrix;

MR: matrixR.Matrix := matrixR.oneMatrix;

-- Parted vectors and matricies

subtype PIndex is Integer range 0..P-1;

Zhs: array (PIndex) of vectorZh.Vector;

MRhs: array (PIndex) of matrixRh.Matrix;

package matrixAhs is new matrix(PIndex, VectorZh);

Ahs: matrixAhs.Matrix;

function convertFromPart(partI: Integer; ind: IndexPart) return IndexZ is

begin

return partI \* (IndexPart'Last - IndexPart'First) + ind;

end;

function sliceVecN is new sliceVec(IndexZ, IndexPart, convertFromPart, vectorZ, vectorZh);

function sliceMatN is new sliceMat(IndexKR, IndexZ, IndexPart, convertFromPart, VectorZ, VectorZh, matrixR, matrixRh);

function newSize(s1: PIndex; s2: IndexPart) return Integer is

begin

return s1 \* s2;

end;

package merge\_p\_i is new merge\_p(matrixAhs, newSize);

A: merge\_p\_i.vector\_o.Vector;

package calc\_p\_i is new calc\_p(IndexPart, vectorZh, IndexT, VectorT, IndexKR, VectorK, matrixK, MatrixRh);

package tasks\_i is new tasks(calc\_p\_i);

use tasks\_i;

tasks: array (0..P-1) of calc\_part;

sems: array (0..P-1) of Suspension\_Object;

begin

for i in PIndex loop

Zhs(i) := sliceVecN(i, Z);

MRhs(i) := sliceMatN(i, MR);

tasks(i).start(sems(i), e, Zhs(i), T, MK, MRhs(i), Ahs(i));

end loop;

for i in PIndex loop

Suspend\_Until\_True(sems(i));

end loop;

A := merge\_p\_i.mergeVectors(Ahs);

merge\_p\_i.vector\_o.print(A);

end main;