МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

## Факультет информационных технологий и робототехники

Кафедра программного обеспечения вычислительной техники

и автоматизированных систем

**Отчет по лабораторной работе № 7-8**

по дисциплине: ”Распределенная и параллельная обработка данных”

на тему: ***”*** ***Построение плана распараллеливания и его реализация”***

Выполнил**:** студент группы 10701214 Зубарев А. А.

Принял**:** проф. Прихожий А.А.

Минск 2017

# Лабораторная работа 7-8. Построение плана распараллеливания и его реализация.

**Цель работы:** научится реализовывать план.

## Задание

Спроектировать программу для реализации плана, разработанного в предыдущих лабораторных работах.

**Теоретическая часть**

Граф задач (task graph) – это ориентированный ациклический взвешенный граф G=(V, E), в котором V – множество узлов, представляющих задачи, E – множество дуг, представляющих передачу данных и отношение предшествования между задачами. Задача определяется как множество инструкций (операторов), выполняемых последовательно на одном процессоре. Соответствующая вершина ni графа метится числом w(ni), описывающим время решения задачи. Дуга (ni, nj) графа метится числом c(i,j), описывающим время передачи данных от задачи ni к задаче nj. Входным называется узел, не имеющий входящих дуг.

Выходным называется узел, не имеющий исходящих дуг. Остальные узлы называются промежуточными. Задача не может начать выполнение не получив данные от родительских задач. Следовательно, условием запуска задачи на выполнение является завершение выполнения всех задач-предшественников на графе задач. При выполнении этого условия задача немедленно запускается на выполнение, следовательно, поведение графа задач в период выполнения является асинхронным. Решение одних задач может происходить параллельно с передачами данных между другими задачами.

Важнейшей характеристикой графа задач является коэффициент «коммуникация/вычисление» (communication-to-computation ratio), определяемый как отношение среднего времени передачи данных от одной задачи к другой к среднему времени решения одной задачи. Время реализации графа задач определяется суммарным весом всех узлов и дуг, входящих в наиболее длинный путь на графе.

**Стратегия планирования «Зануление дуг»**

Стратегия планирования EZ «зануление дуг» стремится сократить длину частично построенного асинхронного плана на каждом шаге планирования путем рассмотрения дуги с максимальным временем передачи данных. Стратегия назначает две задачи, соединенные наиболее «тяжелой» дугой, на один и тот же процессор при условии, что время частичного плана не увеличивается по сравнению с назначением задач на разные процессоры. Если время увеличивается, задачи назначаются на разные подходящие процессоры. Стратегия EZ сначала строит список задач, упорядочивая их в невозрастающем (убывающем) порядке весов, описывающих времена передачи данных. Первая дуга удаляется из списка, а инцидентные узлы-задачи назначаются на один и тот же либо на разные процессоры. Если задачи назначаются на один процессор, дуга зануляется, что интерпретируется как немедленный запуск последующей задачи после завершения предыдущей задачи с нулевым временем обмена данных на одном процессоре. Задачи, назначенные на один процессор, упорядочиваются в соответствии с отношением предшествования и по возрастанию их уровня в графе задач. Процесс планирования заканчивается, когда все задачи назначены на процессоры.

Число шагов работы стратегии меньше числа задач в графе, поскольку рассматриваемое на каждом шаге зануление одной дуги приводит к назначению на процессоры одной или сразу двух задач. Для выбора процессора, на который назначается задача, используется критерий наиболее раннего времени запуска задачи или критерий наиболее короткого во времени частичного плана.

В данной лабораторной работе было взято за основу 6 лабораторная работа. В примере время выполнения задач и передачи данных указано в **сотнях миллисекундах**.

### Код программы

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Threading;

using System.Windows.Forms;

namespace Lab7\_8

{

public partial class Form1 : Form

{

private EdgeZeroingStrategy EZS;

private int lenPlan;

private List<string> StatusPe = new List<string>();

private double time;

public delegate void SetTextCallback(double t);

public delegate void SetRichTextCallback(string s);

private Thread[] poolThreads;

private List<bool> AccessTask = new List<bool>();

public Form1()

{

InitializeComponent();

}

**Планирование**

private void button1\_Click(object sender, EventArgs e)

{

EZS = new EdgeZeroingStrategy();

EZS.File\_Load("start\_param.txt");

EZS.Planning();

ViewResults();

PlanResult();

}

private void ViewResults()

{

ClearView();

// матрица смежности

dataGridView1.ColumnCount = EZS.OperationsCount + 1;

dataGridView1.RowCount = EZS.OperationsCount + 1;

dataGridView1.ColumnHeadersVisible = false;

dataGridView1.RowHeadersVisible = false;

for (int i = 0; i < EZS.OperationsCount + 1; i++)

{

// по вертикали

dataGridView1.Rows[i].Cells[0].Style.BackColor = Color.FromArgb(110, 110, 110);

dataGridView1.Rows[i].Cells[0].Style.ForeColor = Color.FromArgb(255, 255, 255);

dataGridView1.Rows[i].Cells[0].Value = i;

// по горизонтали

dataGridView1.Rows[0].Cells[i].Style.ForeColor = Color.FromArgb(255, 255, 255);

dataGridView1.Rows[0].Cells[i].Style.BackColor = Color.FromArgb(110, 110, 110);

dataGridView1.Rows[0].Cells[i].Value = i;

dataGridView1.Columns[i].Width = 50;

}

dataGridView1.Rows[0].Cells[0].Value = "Operation \\ operation";

for (int i = 0; i < EZS.OperationsCount; i++)

{

for (int k = 0; k < EZS.TasksConnections[i].Length; k++)

{

if (EZS.TasksConnections[i][k] != "0")

{

dataGridView1.Rows[i + 1].Cells[k + 1].Style.BackColor = Color.FromArgb(82, 97, 160);

}

dataGridView1.Rows[i + 1].Cells[k + 1].Value = EZS.TasksConnections[i][k];

}

}

}

private void ClearView()

{

dataGridView1.ColumnCount = 1;

dataGridView1.RowCount = 1;

dataGridView1.ColumnCount = 1;

dataGridView1.RowCount = 1;

richTextBox1.Text = "";

}

private void PlanResult()

{

lenPlan = EZS.FindLengthPlan(EZS.ProcessorsList);

richTextBox1.Text = "time";

List<int> power = new List<int>();

for (int i = 0; i < EZS.ProcessorsList.Count; i++)

{

richTextBox1.Text += "\tPE" + i;

power.Add(0);

}

richTextBox1.Text += "\n";

for (int i = 0; i < lenPlan + 1; i++)

{

richTextBox1.Text += i;

for (int pe = 0; pe < EZS.ProcessorsList.Count; pe++)

{

string name = GetTaskNameOnThisTime(EZS.ProcessorsList, i, pe);

richTextBox1.Text += "\t" + name;

}

richTextBox1.Text += "\n";

}

for (int i = 0; i < EZS.ProcessorsList.Count; i++)

{

for (int k = 0; k < EZS.ProcessorsList[i].Count; k++)

{

power[i] += EZS.ProcessorsList[i][k].TimeCalculation;

}

}

richTextBox1.Text += "\nLoad:";

for (int i = 0; i < EZS.ProcessorsList.Count; i++)

{

double load = Convert.ToDouble(power[i]) / Convert.ToDouble(lenPlan);

richTextBox1.Text += "\t" + load.ToString(String.Format("##.## %"));

}

}

private string GetTaskNameOnThisTime(List<List<EZTask>> procList, int time, int pe)

{

foreach (var task in procList[pe])

{

if (time >= task.StartTime && time <= task.EndTime)

{

return task.Name;

}

}

return " ";

}

private void Clock()

{

while (time \* 10 < lenPlan)

{

lock (this)

{

time += 0.01;

SetTextTime(time);

Thread.Sleep(10);

}

int a = (int)((time \* 10 - Math.Truncate(time \* 10)) \* 10);

// Каждые 100 мс вывод результатов на форму.

// Внимание! Аномалия вывода!

// При выводе, не останавливая все потоки, выводит с задержкой.

// При Debug выводит правильно.

if (a == 0)

{

SetTextStatus(time.ToString(String.Format("#0.#")) + " ");

for (int pe = 0; pe < EZS.ProcessorsList.Count; pe++)

{

lock (StatusPe)

{

if (pe != 0)

{

if (StatusPe[pe] == "Free")

{

SetTextStatus(StatusPe[pe] + "\t\t");

}

else

{

SetTextStatus(StatusPe[pe] + "\t");

}

}

else

{

SetTextStatus(StatusPe[pe] + "\t\t");

}

}

}

SetTextStatus("\n");

}

}

}

public void SetTextTime(double t)

{

if (this.labelTime.InvokeRequired)

{

SetTextCallback d = new SetTextCallback(SetTextTime);

this.Invoke(d, new object[] { t });

}

else

{

this.labelTime.Text = t.ToString(String.Format("##.#"));

}

}

public void SetTextStatus(string s)

{

if (this.richTextBox2.InvokeRequired)

{

SetRichTextCallback d = new SetRichTextCallback(SetTextStatus);

this.Invoke(d, new object[] { s });

}

else

{

this.richTextBox2.Text += s.ToString();

}

}

// Реализация плана

private void button2\_Click(object sender, EventArgs e)

{

// Пул потоков на рассчитанное количество процессоров

poolThreads = new Thread[EZS.ProcessorsList.Count];

// Очистка статусов каждого процессора

StatusPe.Clear();

labelTime.Text = "0";

time = 0; // Начало времени

richTextBox2.Text = "";

// Создание n потоков - процессоров

for (int i = 0; i < EZS.ProcessorsList.Count; i++)

{

poolThreads[i] = new Thread(new ParameterizedThreadStart(Processor));

StatusPe.Add("");

}

// Запуск часов

Thread clock = new Thread(Clock);

clock.Start();

// Запуск потоков - процессоров

for (int i = 0; i < EZS.ProcessorsList.Count; i++)

{

poolThreads[i].Start(i);

}

}

private void Processor(object pe)

{

while (Math.Truncate(time \* 10) < lenPlan)

{

string s = "";

int sleepLength = 0;

int task = -1;

lock (EZS)

{

// Получение имени задачи на текущее время на указанном процессоре

s = GetTaskNameOnThisTime(EZS.ProcessorsList,

(int)Math.Truncate(time \* 10), (int)pe);

lock (StatusPe[(int)pe])

{

if (s == " ")

{

StatusPe[(int)pe] = "Free"; // Свободный процессов

}

else

{

// Процессор занят

StatusPe[(int)pe] = "Work " + s;

sleepLength = EZS.FindEZTaskTimeCalculation(s) \* 100;

task = EZS.FindTask(s);

}

}

}

if (s != " ")

{

Thread.Sleep(sleepLength); // Имитация выполнения задачи

// Запуск имитации передачи данных на другие задачи

// Если таковые имеются

Thread conThread = new Thread(RunConnect);

conThread.Start(task);

}

}

}

// Имитация передачи данных на другие задачи

private void RunConnect(object task)

{

if (EZS.Tasks[(int)task].Connections.Count > 0)

{

// Отправить в отдельный поток передачу данных с указанной задачи

foreach (Connection connection in EZS.Tasks[(int)task].Connections)

{

Thread conThread = new Thread(ConnectThread);

conThread.Name = "connect + " + connection.BeginTaskName + "-" + connection.TaskName;

// Запуск потока с передачей времени передачи данных в мс

conThread.Start(connection.TimeConnection \* 100);

}

}

}

// Имитация передачи данных на одну задачу

private void ConnectThread(object sleep)

{

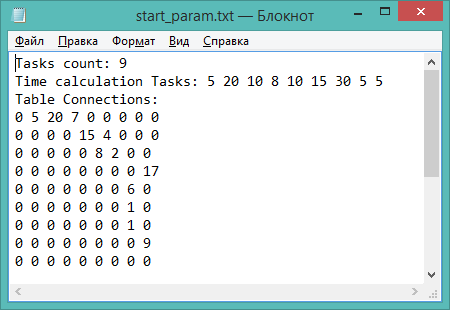
Thread.Sleep((int)sleep);

}

}

}

### Скриншоты результатов



## Вывод

Изучен стратегию минимизации времени длины асинхронного плана.