ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

ФАКУЛТЕТ ПО КОМПЮТЪРНИ СИСТЕМИ И УПРАВЛЕНИЕ

Катедра "Компютърни системи"

СПЕЦИАЛНОСТ: "КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ" - СТЕПЕН БАКАЛАВЪР

ПАРАЛЕЛНО ПРОГРАМИРАНЕ

Упражнение 4

Паралелни алгоритми за обработка на графи

Основни понятия, използвани за графи

Графът се разглежда като съвкупност от върхове (възли) и дъги (ребра). Използва се представяне на съвкупност от обекти и техните връзки. Обикновено върховете съответстват на обектите, дъгите – на връзките между тях.

Математическо определение за граф

Граф G=(V, E) се състои от крайно, непразно множество на върховете V, и множество на ребрата E. n е броя на върховете V, а e – броя на ребрата E. Ако ребрата са представени във вид на подредени двойки (v, w), то графът се нарича ориентиран, v е начало, а w – край на дъгата (v, w). Ако ребрата не са наредени двойки, а множества от два елемента, то граф се нарича неориентиран. (В ориентирания граф може да има ребро (a, a), а в неориентирания – не).

Ако в ориентирания граф G=(V, E), подредената двойка (v, w) принадлежи на E, то казваме, че има ребро то v към w. Ако в множеството на ребрата E на неориентирания граф G=(V, E) има множество [v, w], то считаме, че има ребро между v и w и казваме, че възелът(върха) v е свързан с върха w.

Степен на връх – броя на върховете пряко свързани с върха.

Пълен граф – граф с пълен брой дъги. Степента на всеки връх тогава е n-1 за неориентиран и n за ориентиран, а общия брой дъги e=n*(n-1)/2 за неориентиран и $e=n^2$ за ориентиран.

Граф с тегла – на всеки възел и/или дъга съответства някаква стойност.

Път в граф – списък от върхове (v1, v2, ..., vn), всеки два съседни от които са свързани с дъга. Казваме, че пътят започва от v1 и завършва в vn или, че има път от v1 до vn. Дължината на път в граф без тегла е равна на броя ребра, през които той минава и за пътя (v1, v2, ..., vn) е равна на n-1. За граф с тегла дължината на пътя (v1, v2, ..., vn) е равна на сумата от теглата на ребрата, през които преминава пътя.

Прост път – път, в който няма повтарящи се върхове.

Цикъл – прост път, от поне едно ребро, чиито начален и краен връх съвпадат. В неориентираните графи минималния брой ребра е 3.

Ацикличен граф – граф, който няма цикли.

За неориентирани графи се използват следните понятия:

Свързана компонента на граф G=(V, E) е подграф G1=(V1, E1), така че между всеки два върха от V1 има път, който се състои от ребра е на E1 и няма път между които и да е два върха, единият от които принадлежи на V1, а другият – не принадлежи.

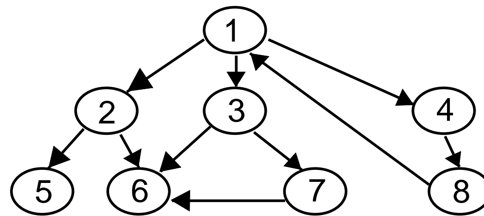
Свързан граф – в него има път между всяка двойка върхове, състои се от една компонента.

Многосвързан граф – състои се от две или повече свързани компоненти.

Представяне на графите в задачите

На възлите на графа се съпоставят номера от 1 до n или имена, които лесно се преобразуват в числа. Начинът за представяне на граф е матрицата на съседство – двумерен масив с размери n*n с числени стойности, в които елемент с индекси v и w има стойност число, която е теглото на дъгата и е 0, когато няма дъга между върховете v и w. При неориентираните графи, на практика, всяка дъга се представя с две стойности, съответстващи на дъга v, w и w, v. Възможно е да се пази само половината матрица, но тъй като се усложнява ненужно реализацията на някой от алгоритмите, се използва цялата матрица.

Нека да разгледаме примерен граф:



фиг.13

Така би се представил чрез използване на матрица на съседство. Реализира се с двумерен масив.

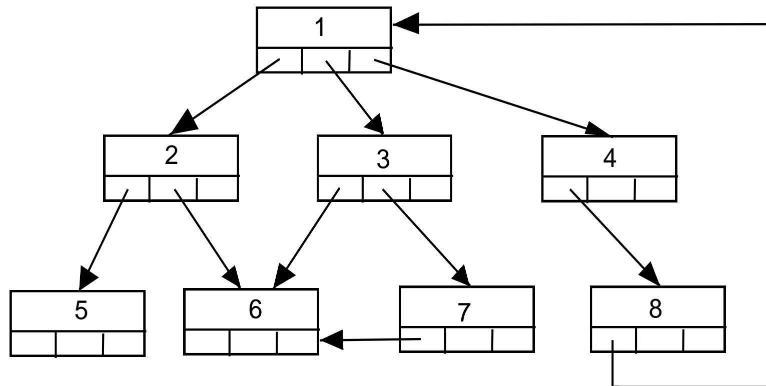
V	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	0	**58**	**5**	**21**	0	0	0	0
2	0	0	0	0	**12**	**7**	0	0
3	0	0	0	0	0	**12**	**9**	0
4	0	0	0	0	0	0	0	**5**
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	**14**	0	0
8	**3**	0	0	0	0	0	0	0

Фиг.1

Друго възможно представяне е чрез динамични структури:

а) ако знаем максималния брой излизащи стрелки от даден възел, може всеки елемент на графа да има компонента - масив от указатели към своите наследници;

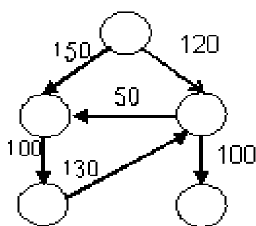
б) ако не знаем максималния брой излизащи стрелки от даден възел, те (излизащите стрелки) се организират в списък.

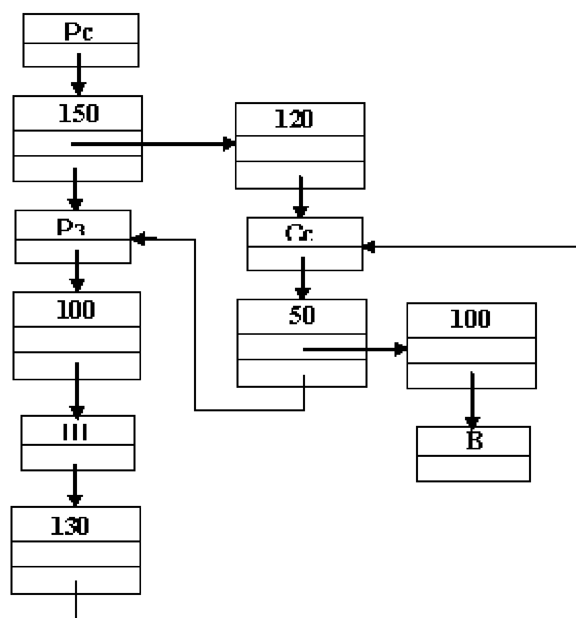


фиг.2

- **Пример**: Нека да разгледаме граф с етикетирани стрелки, описващ връзките и разстоянията между зададени градове.

Графът от фиг. 3а би се представил чрез структурата от фиг. 3б:



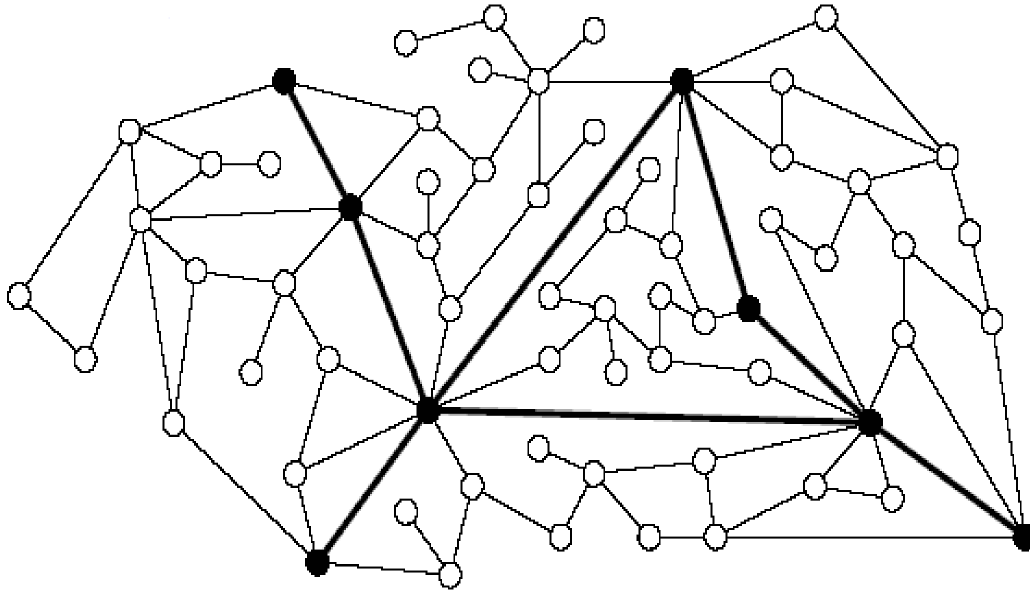


фиг. 3а

фиг.3б

Проектиране на паралелни алгоритми за графи

При паралелизирането на алгоритмите за графи се минава през няколко стъпки. Първата е разделянето на графа с n върха на r подграфа, както е показано на фиг. 4. Всеки подграф има n/r върха и (n/r)1/2 гранични върха. Тази стъпка отнема O(n) време.

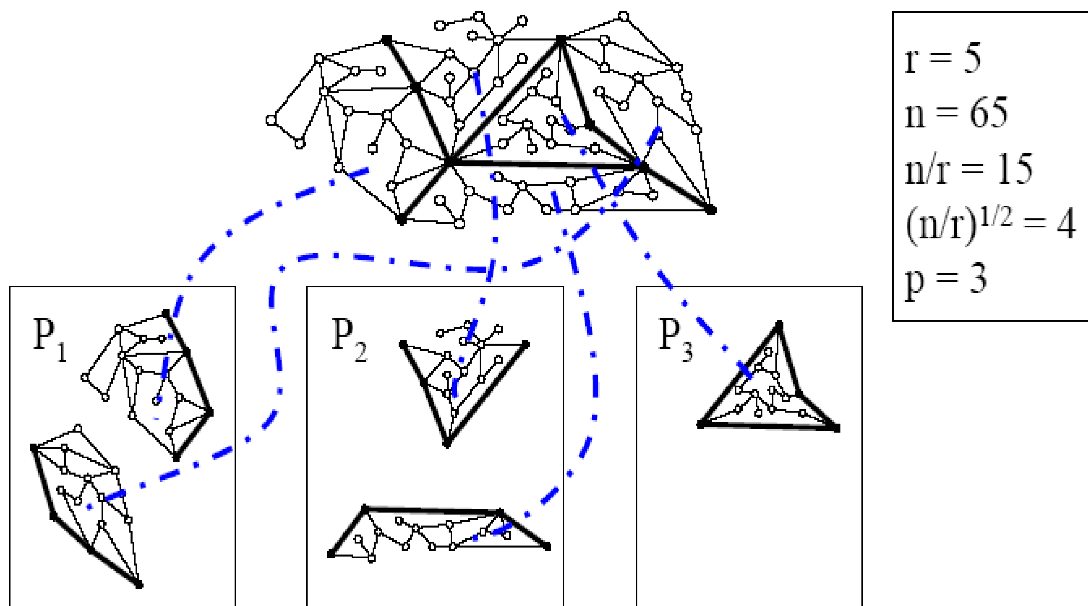


фиг. 4 Разделяне на графа на подграфи

Този етап от проектирането на паралелния алгоритъм съответства на първия етап - разделяне (partitioning). Разделянето на графа става на приблизително равни части и се определя от броя на процесорите и броя на възлите в графа. Необходимо е, за да може да се разпаралели обработката на графа.

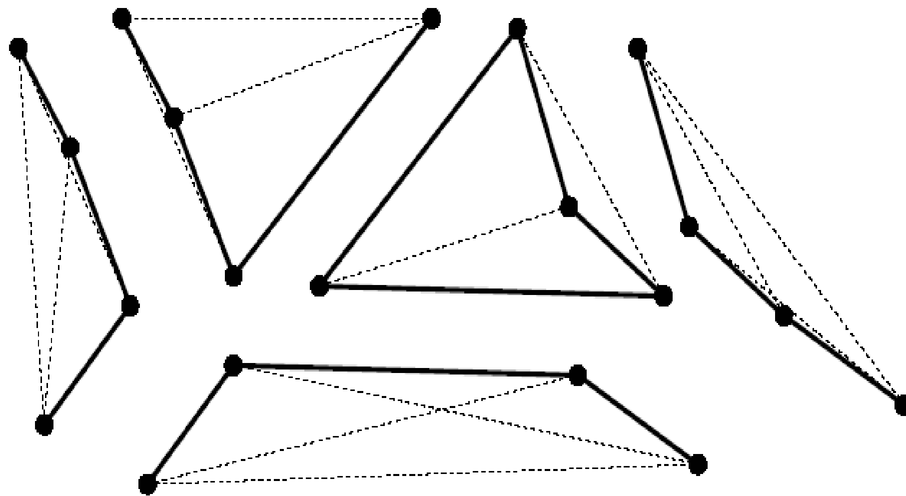
На следващата стъпка графът се зарежда от главния процесор (master, root). Подграфите, които са r на брой, са присвоени случайно на p подчинени процесори (worker). Целта е разпределянето на вече разделения граф на процесорите, така че всеки от тях да извършва паралелна обработка и накрая да изпрати резултата към главния процесор, който да обработи отделните решения и изведе крайния резултат.

Резултатите, получени при едно такова разделяне са много добри и дават възможност работата по обработката на графа да се раздели на независими части и да се осигури паралелна обработка.



фиг. 5 Разпределяне на отделните подграфи на всеки процесор

Стъпка 3: Всеки процесор изчислява самостоятелно за собствения граф търсената величина - най-кратък път, връх с минимално тегло или др.



фиг. 6 Всеки процесор работи над собствения си подграф

И най-накрая всеки изпраща решението към главния процес, който избира най-доброто решение и прекратява изпълнението на работните процеси (worker).

Задача за търговския пътник (Traveling Salesman Problem-TSP)

Хора от различни професии (пощальони, разносвачи на стоки по домовете, търговци и т.н.) често са изправени пред следната задача: те трябва да тръгнат от точка, съответна на работното им място (пощенска станция, магазин, склад или офис на фирмата), да посетят няколко други точки, където са разположени клиентите им и да се върнат в началната точка на обиколката. Нека точките, през които трябва да мине обиколката са означени с числата от 1 до N, като началната точка е означена с 1. За всеки две точки, които трябва да бъдат посетени (включително и началната точка на обиколката) е известно цяло число – разстояние между двете точки (време за изминаване на това разстояние или някаква друга цена за изминаването му). Няма съмнение, че колкото по-кратка (по-бърза, по-евтина) е обиколката, толкова по-добре. За съжаление, добре известно е, че да се намери най-добрата възможна обиколка не е никак лесно, ако броят на местата, които трябва да бъдат посетени е достатъчно голям.

Формулировка на задачата:
Дадени са градове и разстоянията между тях. Разстоянията са присвоени на всяко ребро, свързващо два града. Търговският пътник трябва да мине през всичките градове, но така че да измине най-малкия възможен път, определен от сумата на теглата на ребрата на графа (теглата са разстоянията между градовете). Най-накрая търговският пътник трябва да се върне в началната точка (там откъдето е тръгнал).

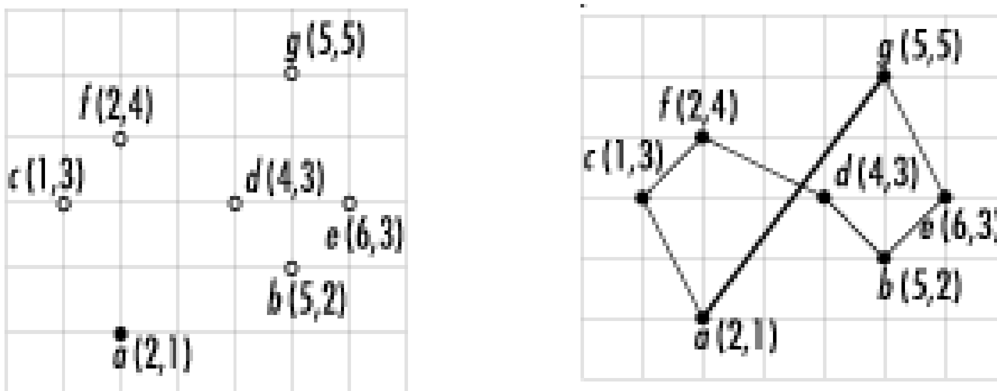
Целта на задачата е да се намери последователност от градове, така че да се намали разстоянието за пътуване.

Последователно решение
Най-простото решение на алгоритъма е да се обходят всички възможни пътища, да им се изчисли дължината и да се избере този с най-малка дължина. Но това е и най-бавното решение и работи само за ограничен брой възли. Той завършва за полиномно време и ако броя на възлите е n, то след n! итерации се достига крайното решение. Задачата за търговския пътник е специален тип, наречени NP-пълни, които завършват за експоненциално време. Съществуват и други приблизителни методи за решение - евристичен, генетичен и др., които не дават точното решение, но резултати, близки до точното.
Евристичният алгоритъм се опитва от текущия връх да отиде във всеки връх, който не е посетен и от този, в който е отишъл отново се опитва да отиде във всеки непосетен и т.н.
Ще решим задачата чрез пълно изчерпване. Избира се произволен връх i и тръгвайки от него се построяват всички последователни маршрути. Дължината curSum на маршрута, построен до момента, се променя в тялото на рекурсията. Когато се намери нов път, обхождащ всички върхове на графа само по веднъж, се проверява дали дължината му е по-малка от най-малката, намерена до момента (съхранява се в променлива minSum). Като тук се прави следното подобрение: ако на някоя стъпка дължината на curSum на пътя, който се строи в момента стане по-голяма от minSum, се прекъсва строенето на текущия маршрут.

Структури от данни:
Ако искаме да разрешим задачата на един компютър, трябва да се намери начин да се представи тегловния, неориентиран граф. Матрицата на съседство е избраната структура от данни за това приложение, защото позволява константно време за достъп до всеки връх и не консумира повече памет, отколкото се изисква за съхранението на решението. Елементът (i,j) е теглото на реброто от връх i до връх j, а на несъществуващите ребра се присвоява безкрайност (изключително високо число) или 0.
Задачата се реализира като входните данни се четат от файл и се записват дължините на теглата в двумерен масив. Файлът се генерира предварително на случаен принцип или се въвеждат от потребителя дължините на теглата.
Използва се и едномерен масив U с големина MAXN, в който се маркират елементите, който са посетени.
Едно пълно обхождане на графа се съхранява в масива cycle, а минималния път - в масив minCycle. Решението от последователни възли, описващи минималния път в графа, се записва в едномерен масив - minCycle.

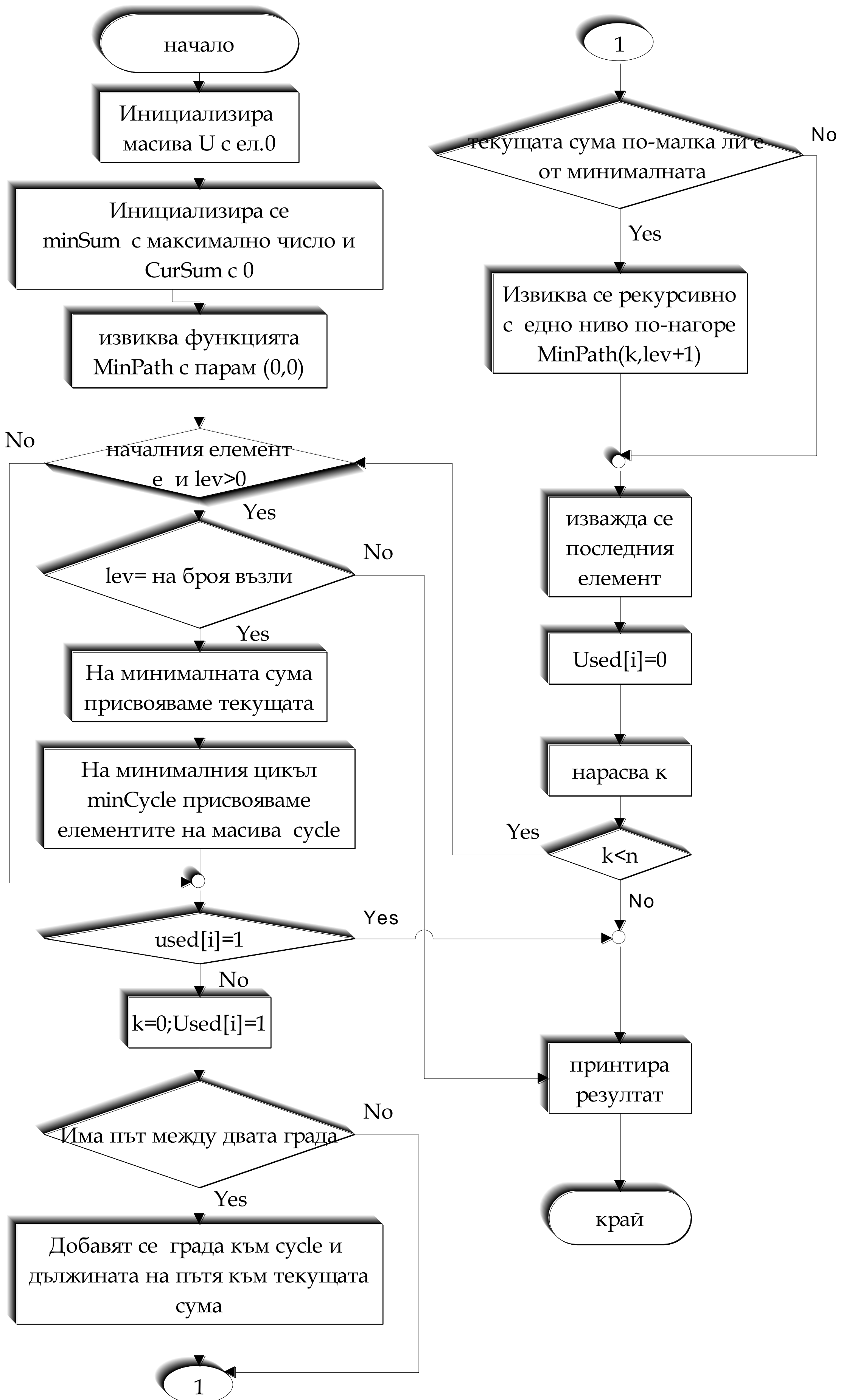
Име на структурата	Тип елементи	големина
A	int	MAXN*MAXN
U	char	MAXN
minCycle	unsigned	MAXN
cycle	unsigned	MAXN

Дължината на пътя се съхранява в променливите curSum и minSum.



фиг. 7 Решение на задачата за търговския пътник

Блок схема:


начало
Инициализира масива U с ел.0
Инициализира се minSum с максимално число и CurSum с 0
извиква функцията MinPath с парам (0,0)
No
началния елемент е и lev>0
Yes
lev= на броя възли
No
Yes
На минималната сума присвояваме текущата
На минималния цикъл minCycle присвояваме елементите на масива cycle
used[i]=1
Yes
No
k=0;Used[i]=1
Има път между двата града
No
Yes
Добавят се града към cycle и дължината на пътя към текущата сума
1
1
текущата сума по-малка ли е от минималната
No
Yes
Извиква се рекурсивно с едно ниво по-нагоре MinPath(k,lev+1)
изважда се последния елемент
Used[i]=0
нарасва к
Yes
k<n
No
принтира резултат
край

Паралелно решение

Проектиране на алгоритъм за търговския пътник
Първата стъпка е да се определи дали да се избере разделяне на областта или функционално разделяне. В този случай, изборът е ясен. Алгоритъмът изпълнява едни и същи твърдения и не може да има функционално разделяне. Така че лесно се изпълнява разделяне на областта.

Паралелният алгоритъм се състои в разделянето на 2 основни вида процеса: координатор и подчинен (worker). Координаторът се извиква от процес с ранг 0, а всички останали процеси са подчинени (броят им се определя от броят на процесорите, на които се стартира алгоритъма).
Координаторът разпределя масива с дължини на разстояния на равни части и ги разпространява към всички процесори. С това се изпълнява първата стъпка от проектирането на паралелния алгоритъм. Координаторът получава съобщение от работниците и в зависимост от вида на съобщението, той изпраща следните съобщения:
От страна на координатора:
1. Ако е получил съобщение с етикет, че е намерен по-добър път и дължината му е по-малка от текущия най-добър път, обновява най-късия път с намерения по-добър и изпраща дължината му на всички с етикет да подновят най-добрия път.
2. Ако е получен етикет за поставяне на частичен път в опашката, но има чакащи процеси, пътя се изпраща на един от чакащите процеси, в противен случай се поставя частичния път в опашката, дотам, докъдето са посетени.
3. Ако опашката е празна, чака и ако е стигнал последния процес, изпраща съобщение за край към всички.
Подчинените процеси, ако получат по-добър път, го заменят с текущия и изчисляват разстоянието между градовете.

- Има схема координатор/работник
  - Координатора съхранява поделените структури данни (главната опашка и текущия най-добър път);
  - Подчинените (Workers) разширяват с 1 частичен път по това време получава се нещо като клиент/сървър приложение-клиента изпраща заявки, сървъра отговаря на заявките.

Съгласно тези определения се извършва следната MPI комуникация:

1.*<u>От работника към координатора</u>*
- Заявка за нов частичен път за изследване;
- Добавяне на нов частичен път в опашката;
- Подновява координатора, ако е намерен нов по-добър път.

2.*<u>От координатора към работника</u>*
- Нов частичен път за изследване
- Край на изчисленията (няма повече частични пътища)
- Нова най-добра дължина на пътя ако съществува

Структури от данни:

Клас Path

За съхраняването на текущия път се използва класа Path с три полета: Първото е текущата дължина на частичния път, второто е масив, пермутация на всички градове с големина MAXCITIES, последното поле показва номера на градовете в частичния път.

City[0] …city[visited-1] - текущия частичен път
City[visited]….city[NumCities] - градовете, които все още не са в пътя

Класът съдържа и три функции:

Path() - конструктор на класа
Set - присвоява стойности на полета visited, length, city
Print()-принтира получения път

Изброен списък от етикети, които подпомагат комуникацията между координатора и работниците. Използват се за синхронизация на работата и разграничаване на отделните съобщения.

Структура Msg_t - служи за съхраняване на дължината, броя на градовете на частичния път и масив от индекси на градовете. Съдържа 3 полета - length, city, visited.

Частичните пътища се натрупват в списъчна структура, за която са реализирани методите по проверка дали списъкът е празен, добавяне на елемент в списъка, премахване на елемент.
Декларират се два класа:
class ListElement с полета:

List Element () - конструктор, инициализира елементите на списъка;
List Element *next - следващия елемент в списъка;
int key - ключ определящ приоритета за сортиран списък;
void *item - указател към елемент от списъка;

class List - съдържа конструктор и деструктур и 3 метода-IsEmpty, Inset, Remove

Данните се четат от файл и се записват в масив с MaxCities*MaxCities на брой елемента, които се натрупват в Dist. За него се заделя динамично памет, в зависимост от броя градове NumCities.

В waiting се съхранява броя на чакащите процеси.
Използва се полето tag в съобщенията:

MPI_Send(buf, count, datatype, dest, tag, comm)
MPI_Recv(buf, count, datatype, source, tag, comm, &status)

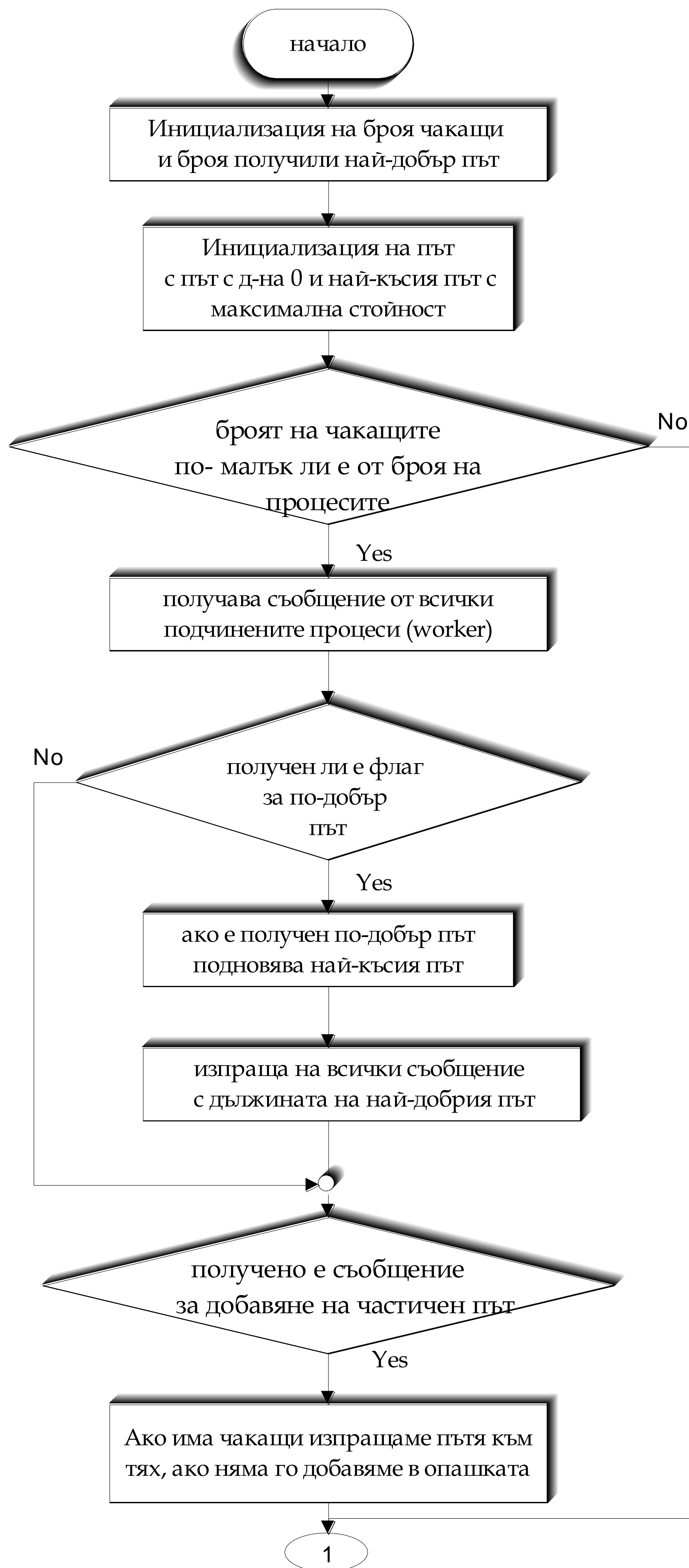
Дефинира се набор от етикети:
PUT_PATH, GET_PATH, BEST_PATH, DONE_TAG, GET_PATH_TAG, UPDATE_BEST_PATH_TAG, REPLY_PATH_TAG
Статусът е запис с полета:

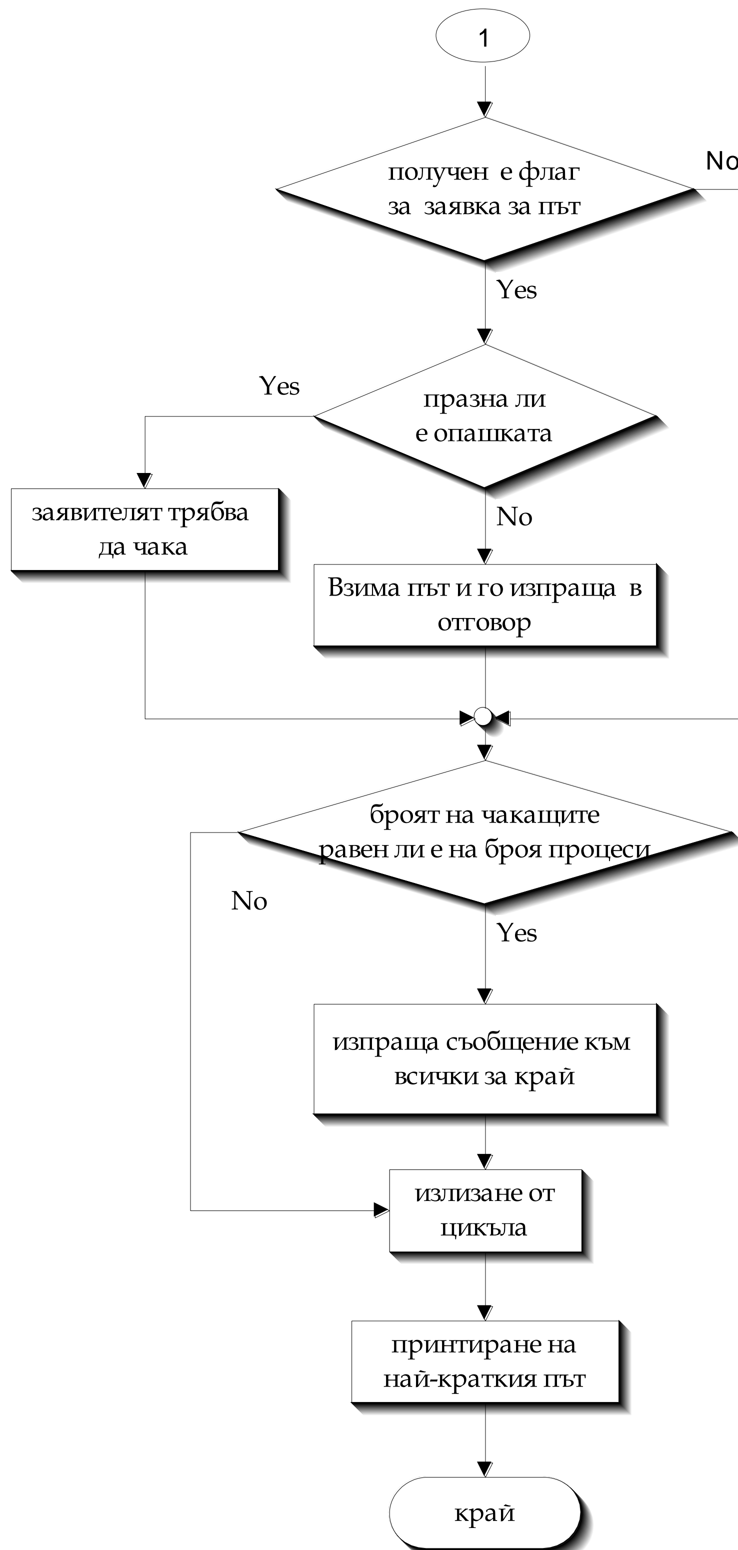
- MPI_SOURCE
- MPI_TAG

Така в MPI_Recv(), когато се запише MPI_ANY_TAG, MPI_ANY_SOURCE може да се намери етикета и източника от статус полето.

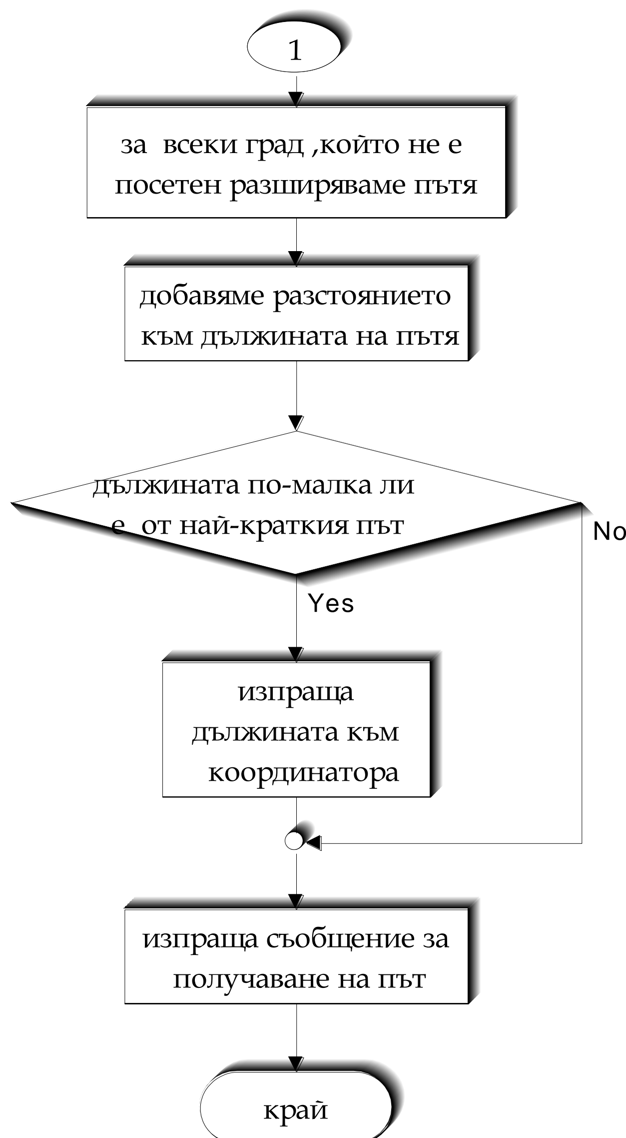
Блок схеми:

Блок схема на функция Coordinator()




1
получен е флаг
за заявка за път
No
Yes
Yes
празна ли
е опашката
заявителят трябва
да чака
No
Взима път и го изпраща в
отговор
броят на чакащите
равен ли е на броя процеси
No
Yes
изпраща съобщение към
всички за край
излизане от
цикъла
принтиране на
най-краткия път
край

Блок схема на функция Worker()

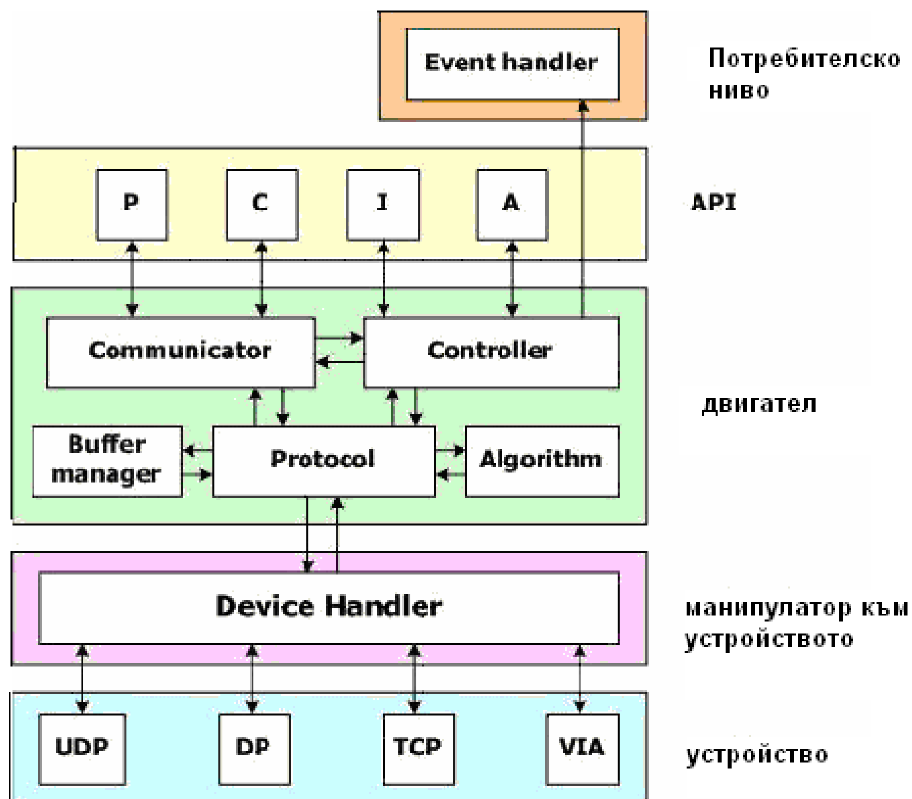


Програмна реализация

Платформа и средства за разработка

MPICH е преносима имплементация на спецификацията MPI за широко разнообразие от паралелни изчисления в разпределена среда. Поддържа основна комуникация от точка в точка. Разработва комуникационна библиотека, която се подчинява на MPI стандарта и предлага толеранс в грешките. MPICH не съдържа само MPI библиотеката, а съдържа и програмно обкръжение за работа с MPI програми. Програмното обкръжение включва стартиращ механизъм и профилиращи библиотеки за изучаване на производителността на MPI програма.

Архитектура на MPICH:



фиг.7

Избор на програмна среда

MPI е библиотека, капсулираща възможността за създаване на паралелни програми чрез използване на езиците "*C*" и "*Fortran77*". Стандартно тези езици не поддържат средства за подобен вид програмиране. Ето защо различните източници, поддържащи тези езици разработват различни способи за това. Това води до появата на редица приложения, които не са преносими от една платформа към друга.
Стандартът MPI е разработен, за да елиминира тези проблеми. Това е библиотека, която се използва от програми на C и Fortran, базира се на стандартните услуги на операционните системи и дава възможност за създаването на паралелни процеси и комуникацията помежду им. MPI позволява разработката на паралелни приложения, които работят успешно на повечето паралелни архитектури. При дизайна на MPI са заимствани идеи от повечето съвременни водещи фирми в областта на компютърната техника (**IBM, Intel, TMC, Cray, Convex** и др.), автори на други библиотеки за паралелно програмиране (**PVM, Linda** и др.) и редица софтуерни специалисти.

a) MPI е единствената библиотека за паралелно програмиране, която се третира като стандартна. Поддържа се за почти всички платформи и на практика е заместила всички подобни библиотеки.
b) Програмите написани с MPI не изискват модифициране при прехвърлянето си от една платформа към друга – поддържаща MPI.
c) Версиите на библиотеката са специфични за дадена платформа и позволяват оптимизирането на кода за нея.
d) MPI се състои от над 115 метода.
e) MPI има много реализации.

Използвани функции

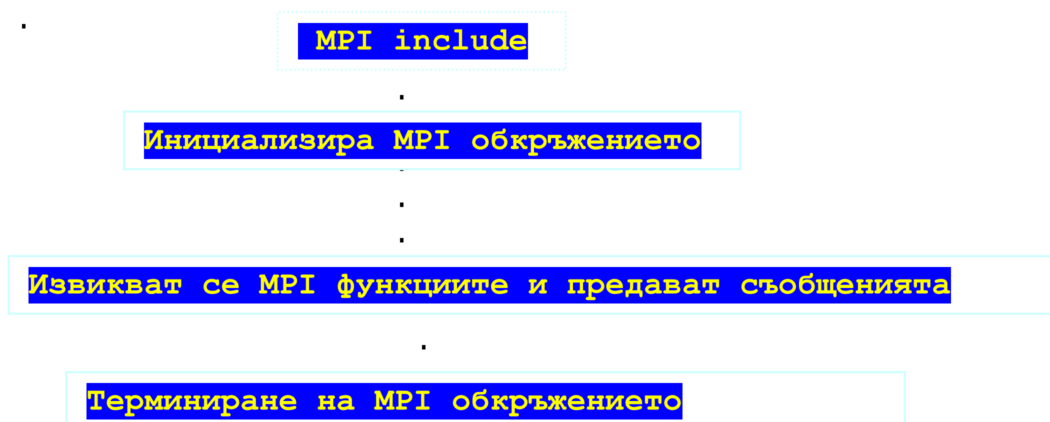
Всички програми, използващи MPI, трябва да включат заглавен файл за достъп до MPI функциите.

C Header File	Fortran Header File
#include "mpi.h"	include 'mpif.h'

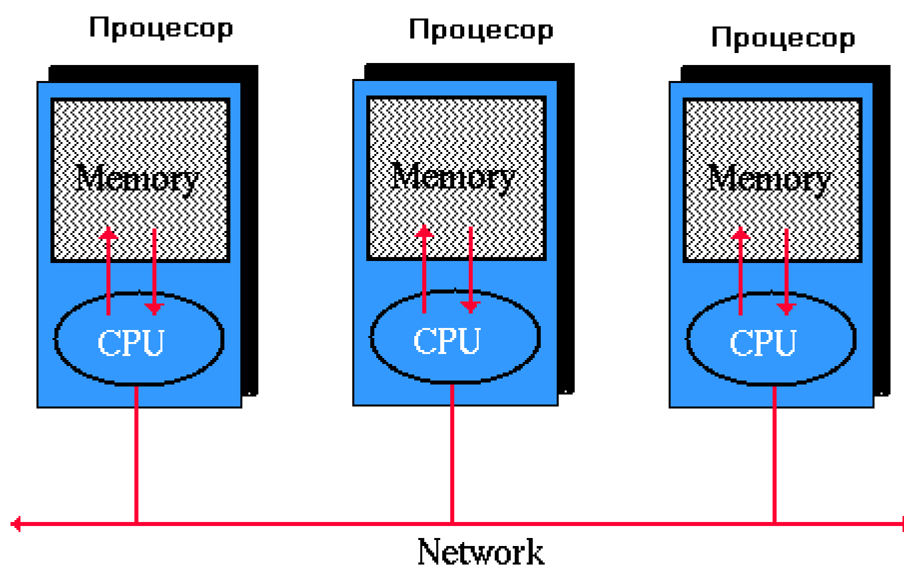
Всички MPI функции се именуват по въведена конвенция и връщат специфичен резултат:

C Binding	
Format:	rc = MPI_Xxxxx(parameter, ...)
Example:	rc = MPI_Bsend(&buf,count,type,dest,tag,comm)
Error code:	Returned as "rc". MPI_SUCCESS if successful

Структура на MPI програма



Message Passing Парадигма
Всеки процес има своя собствена локална памет, която може да бъде достигната директно от собствения процесор (CPU). Трансфера на данни от един процесор към друг е представена през мрежата. Разликата е в поделената памет, която позволява множество процесори директно да достигат един и същи ресурс памет през шината.



. фиг. 8 Система на разпределената памет

Предаване на съобщенията
Метода, по който са копирани данните от паметта на един процесор в паметта на друг процесор. В разпределените системи, данните се изпращат основно като пакети от информация през мрежата от един процесор към друг. Съобщението може да съдържа един или повече пакети и обикновено включва рутираща и/или друга информация.

Процеси
Процесът е набор от изпълними инструкции (програми), които се изпълняват на процесора. Един или повече процеси могат да се изпълнят на един и същи процесор. В системите с предаване на съобщения, всички процеси комуникират един с друг, като изпращат съобщения - дори и да се изпълняват на един и същи процесор. От съображения за ефективност, системите с предаване на съобщения асоциират само един процес на процесор.

Библиотеки за предаване на съобщения
Колекция от функции, които са вградени в кода на приложението, за да извършат send, receive и други операции за предаване на съобщения.

Send/Receive
Предаването на съобщения включва трансфера на данни от един процес (send) до друг процес (receive). Изисква съгласуване както на процеса на изпращане, така и на процеса на получаване на съобщения. Send операцията обикновено изисква изпращащия процес да определи разположението, големината, типа и предназначението на данните. Операцията Receive трябва да отговаря на съответстващата и send операция.

Предаване на съобщенията

Процес A

памет

data

send(data)

мрежа

Процес B

памет

data

send(data)

Фиг.9

Комуникатори и групи

MPI Използва комуникатори и групи, за да определи съвкупността от процеси, които могат да комуникират един със друг. Повечето MPI функции изискват да се подаде комуникатор като аргумент. MPI_COMM_WORLD е предефиниран комуникатор, който се използва по подразбиране и най-често той се използва, освен ако не се създаде собствен комуникатор.

MPI_COMM_WORLD

6 2 7 5 8 0 1 9 3 4

Фиг.10 Комуникатора, обединяващ общите процеси в група

Една MPI програма се състои от процеси, изпълняващи самостоятелно собствен код. Кодът се изпълнява от всеки процес. Процесите комуникират чрез MPI комуникационни примитиви. Всеки процес се изпълнява на собствено адресно пространство.

MPI (Message-Passing Interface)

- Стандартизиран набор от операции и типови данни, които подсигуряват образец на предаването на съобщенията
- SPMD(Single Program Multiple Data)

Всички процеси изпълняват собствено копие на една и съща програма. Типове данни в MPI:

MPI C data types	
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int

MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	8 binary digits
MPI_PACKED	data packed or unpacked with MPI_Pack()/ MPI_Unpack

Описание на функциите:
Основни функции, предоставени от MPI средата:

- MPI_Init
  -инициализира MPI библиотеката
  -инициализира MPI_COMM_WORLD - колекция от процеси който са стартирани
  -взима копие от параметрите на комадния ред
  -Трябва да бъде извикан преди всяка MPI функция

MPI_Init (*argc,*argv)

- MPI_Comm_size
  определя броя на процесите който са стартирани

MPI_Comm_size (comm,*size)
comm. - комуникатора асоциран с групата
size-броя на процесите в групата

- MPI_Comm_rank
  определя номера или идентификатора на даден процес(0 до size-1), където size е броят на процесите

MPI_Comm_rank (comm,*rank)
comm - комуникатора
rank - ранга или IDE на процеса

- MPI_Send
  -изпраща съобщение до друг процес
  -буфер съдържащ съобщението
  -типа на данните в съобщението
  -получателя на съобщението
  -ранга на процеса получател

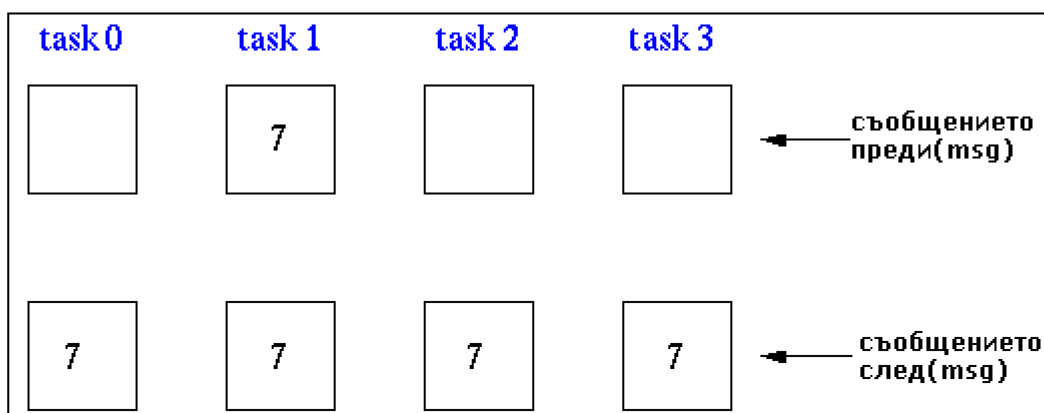
MPI_Send (*buf,count,datatype,dest,tag,comm)

Основна блокираща изпращаща операция. Функцията завършва само когато буфера в изпращащата задача се освободи за повторно използване.

- MPI_Bcast

Изпраща съобщение от процес с ранг 0 до всички останали процеси в групата.

MPI_Bcast (*buffer, count, datatype, root, comm)
buffer - съобщението, което ще предаваме
count - големината на съобщението
datatype - типа на елементите



фиг. 11 Разпределението на заданията след операцията MPI_Bcast

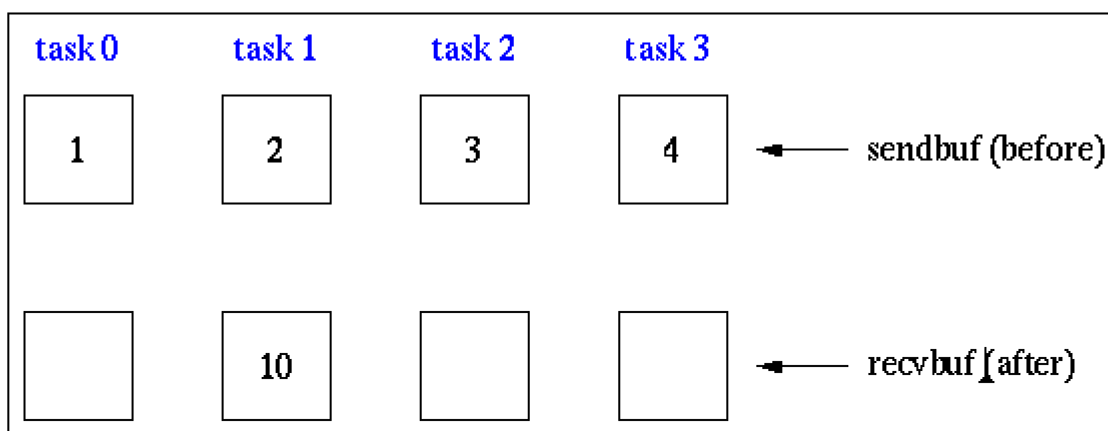
- MPI_Gather

Събира различните съобщения от всяка задача в групата до отделен получател.

MPI_Gather (*sendbuf,sendcnt,sendtype,*recvbuf, recvcount,recvtype,root,comm)

- MPI_Reduce

MPI_Reduce (*sendbuf,*recvbuf,count,datatype,op,root,comm)
Извършва операция събиране и поставя резултата в една от задачите.



Фиг. 12 Състояние на заданията след операцията MPI_Reduce

Групи и комуникатори:
Групата е подреден набор от процеси. Комуникаторът събира група от процеси, които могат да комуникират един със друг.

Виртуални топологии:
MPI предоставя средства да подреждане на процесите от една група в топология. Топологиите са виртуални, може да няма връзка между физическата структура на паралелната машина и топологията на процесите. В примера използвам топология Graph. Тази структура е много удобна за използване при представянето на графа, тъй като идеално съответства на структурата на топологията. Виртуалните топологии се строят върху комуникатори и групи.
Прави нов комуникатор, за който информация за топологията може да бъде присвоена.

MPI_Graph_create (comm_old, nnodes, *index[],*edges[], reorder,*comm_graph)
comm_old - входния комуникатор
nnodes - броя на възлите в графа
index - масив от цели числа, описващ степените(номерата) на възлите
Reorder - булева стойност, показваща дали ще се пренареждат номерата на процесите или не
*comm_graph - новия комуникатор

Задача за търговския пътник

Последователна версия:

void PrintCycle(void)

Функция, която показва на екрана намерения минимален път в графа и неговата дължина.

void MinPath(unsigned i,unsigned lev)
Рекурсивна функция, която обхожда графа и намира Хамилтонов цикъл и когато го намери проверява дали дължината му е по-малка от намерената до момента.

Паралелна версия:
В програмата се използват следните функции:

Path::Path ()
Конструктор на класа, който инициализира полетата length (дължината на пътя с 0), visited (броя на посетените градове с 1) и масива с индекси на градовете.

void Path::Set (int len, int *cit, int vis)
Метод на класа, който задава стойности за полетата на класа.

С първия параметър се задава дължина на пътя, вторият е указател към масив от индекси на градове, третият параметър задава стойност на броя на посетените градове.

void Path::Print()
Целта му е да изведе на екрана пътя, състоящ се от посетени върхове и дължината на този път. Така се отпечатва и най-краткия път.

void Fill_Dist(void)
Тук се чете от файл матрицата с разстоянията между градовете и техния брой. Заделя се памет за динамичен едномерен масив, в който се съхраняват и с функцията MPI_Bcast се разпространява броя на градовете към всички. Със същата функция се подава и масива. Главният процес показва масива на екрана.

void Coordinator ()
Координира работата на подчинените процеси и им изпраща съобщения в зависимост от това по добър път ли е намерен или е направена заявка за подаване на нов частичен път за изследване. Той изпраща и съобщение за край при намирането на минимален път. Накрая разпечатва намерения път.

void Worker ()
Добавя нови дължини и строи минималния път докато не получи съобщението за край от координатора. Извиква се от всеки от процесите, които се изпълняват на компютрите толкова на брой пъти, колкото са процесите.

void * List::Remove(int *keyPtr)
Функция, която премахва елемент от списъка с определен ключ.

List::Insert(void *item, int sortKey)
Добавя елемент преди елемент с по-голяма стойност. По този начин списъкът се поддържа сортиран.

bool List::IsEmpty()
Проверява дали списъкът е празен и връща истина или лъжа**.**

Анализ на производителността

От получените резултати може да се направят следните изчисления:

- Ускорение:

Определянето на производителността, от архитектурна гледна точка, се предлага да се извършва по простата формула:

$$S=T_1/T_P$$

където:
- S е коефициент на изменение на бързодействието;
- T1 е времето за решаване на дадена задача на едно-процесорен компютър;

- Tp е времето за решаване на същата задача на паралелен компютър.
- Ефективността при паралелизма се дефинира като отношението на времето за паралелно изпълнение на дадена програма:

Коефициентът **E** показва средното натоварване на всички процесори, включени в паралелния компютър. Колкото този коефициент е по-близък до 1, толкова повече процесори работят през цялото време на решаване на задачата и обратно:

$$En=Sn/p$$

Експериментални данни за изпълнение

Входни данни

Като входни данни се въвеждат числа във файлове. На първият ред от входния файл е зададено число, което показва броя на градовете.

12											
0	41	8466	6334	6498	9168	5723	1477	9356	6960	4462	5705
41	0	8143	3279	6826	9961	491	2995	1941	4827	5436	2388
8466	8143	0	4603	3902	153	292	2381	7420	8715	9717	9894
6334	3279	4603	0	5447	1724	4770	1537	1869	9911	5665	6297
6498	6826	3902	5447	0	7034	9894	8701	3809	1319	330	7672
9168	9961	153	1724	7034	0	4664	5140	7711	8251	6868	5545
5723	491	292	4770	9894	4664	0	7642	2659	2754	35	2858
1477	2995	2381	1537	8701	5140	7642	0	8723	9741	7527	778
9356	1941	7420	1869	3809	7711	2659	8723	0	2315	3035	2188
6960	4827	8715	9911	1319	8251	2754	9741	2315	0	1842	288
4462	5436	9717	5665	330	6868	35	7527	3035	1842	0	103
5705	2388	9894	6297	7672	5545	2858	778	2188	288	103	0

Изходни данни

На изхода получава пътя като последователност от възли и дължината на най-краткия път, показва се и комуникацията при стартирането на процесите и при тяхното завършване чрез отпечатване на съобщения.