КЛАСИФИЦИРАНЕ НА ДОКУМЕНТИ

DOCUMENT CLASSIFICATION

©BOROVSKA

WWW – млн. текстови документи

- *Автоматизирани търсачки (search engines) откриват съответната info*
- *За представяне на съдържанието на документа се използва вектор*
- *Всяко измерение на вектора представя съответствието между документа и концепцията, която може да бъде във формата на дума или фраза*

©BOROVSKA

ПАРАЛЕЛНА ПРОГРАМА

- *Чете каталог от ключови думи*
- *Прави селекция на множество текстови документи*
- *Чете документите*
- *Генерира вектор за всеки документ*
- *Записва векторите на документите*
- *Функционална декомпозиция*
- *Стил manager/worker*

Използвани MPI функции:

- **MPI_Irecv** – за инициализиране на неблокиращо приемане (nonblocking receive)
- **MPI_Isend** - за инициализиране на неблокиращо предаване (nonblocking send)
- **MPI_Iwait** – изчакване да завърши неблокираща комуникация
- **MPI_Probe** – проверка за входящо съобщение
- **MPI_Get_count** – определяне на дължината на съобщението
- **MPI_Testsome** – връща информация за всички завършили неблокиращи комуникации

ПАРАЛЕЛЕН АЛГОРИТЪМ

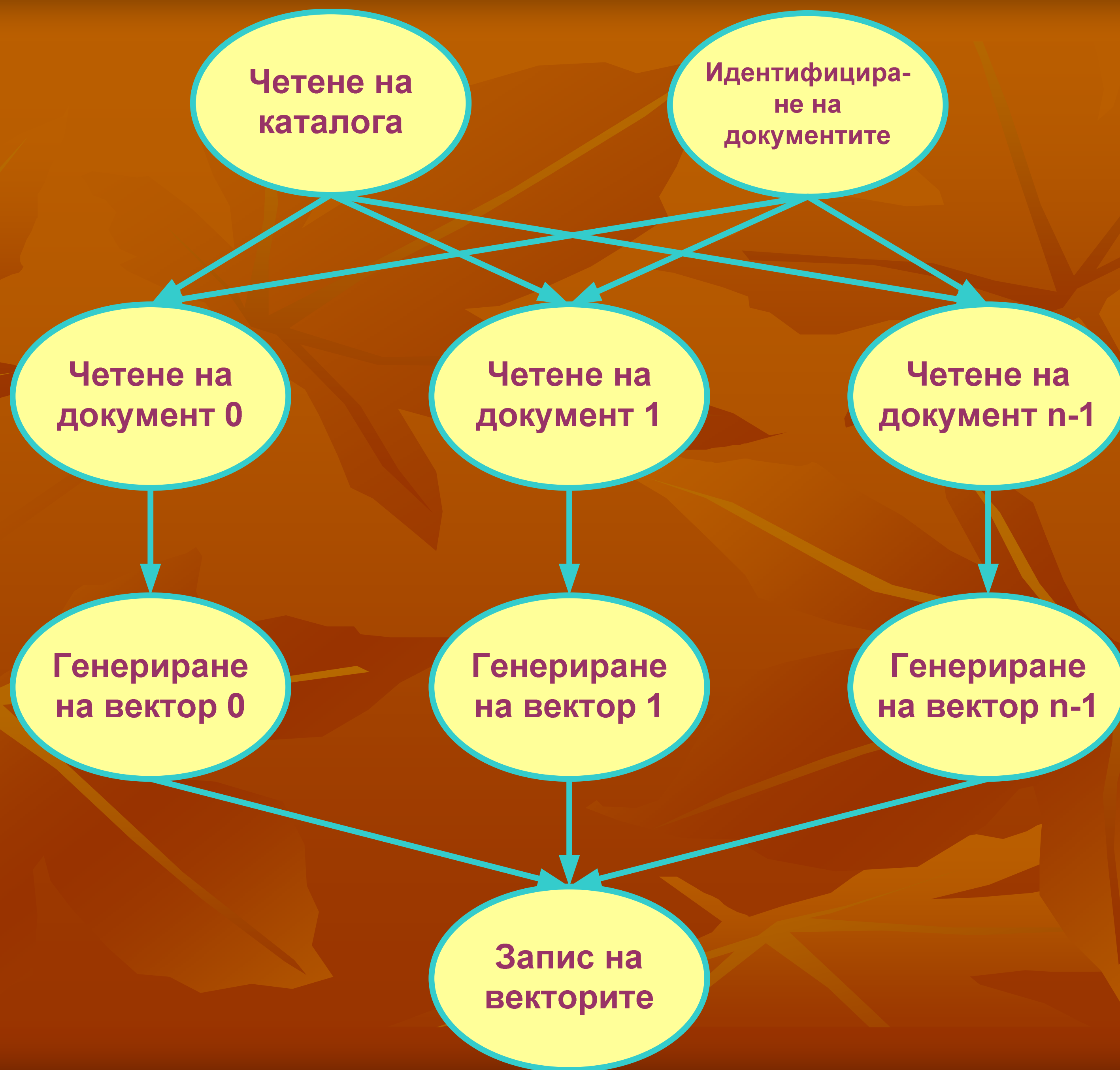
- Програмата чете каталога
- Търси в каталожната структура текстови файлове (.html, .tex, txt)
- Отваря файловете, чете съдържанието, генерира профилиращ вектор, показващ колко пъти текстовият документ съдържа всяка дума от каталога
- Записва файл, съдържащ профилиращите вектори за всеки от прегледаните текстови файлове

ГРАФ НА ЗАДАЧИТЕ НА ПРИЛОЖЕНИЕТО ЗА КЛАСИФИЦИРАНЕ НА ДОКУМЕНТИТЕ

ФУНКЦИОНАЛНА ДЕКОМПОЗИЦИЯ И КОМУНИКАЦИЯ

- ***Четенето на каталога и идентифицирането на документите може да се реализира паралелно***
- ***Четенето на документите и генерирането на профилиращите вектори консумират по-голямата част от времето***
- ***Създават се две задачи за всеки документ – едната чете файла с документа, а другата – генерира профилиращия вектор***

©BOROVSKA



©BOROVSKA

АГЛОМЕРАЦИЯ И МАПИНГ

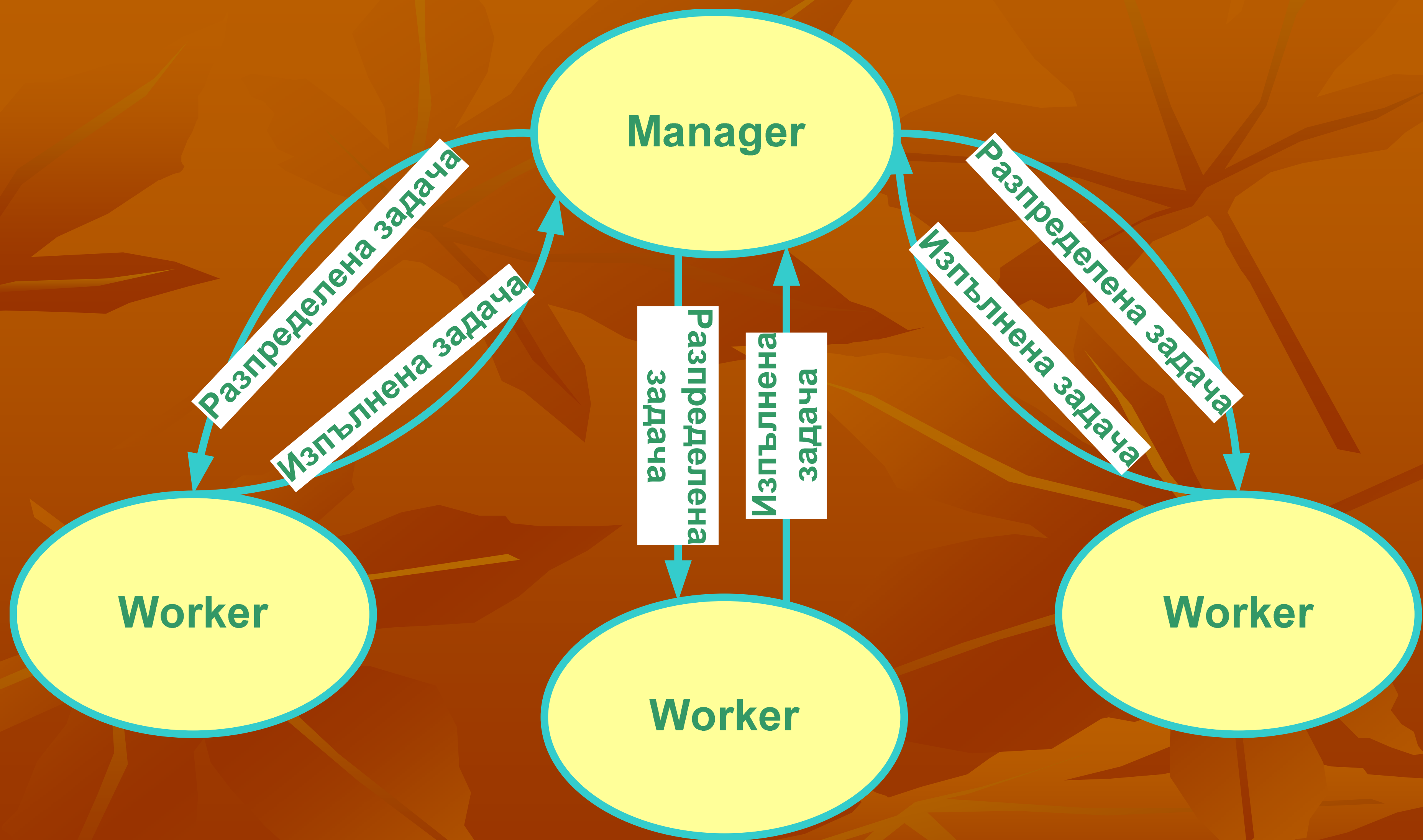
- ***Броят на задачите не е известен при компилацията***
- ***Задачите не комуникират една с друга***
  - ***Работният товар** на задачите за обработка на документите **варира в широки граници** – документите се различават*
- ***Mapping – в хода на изпълнението на програмата (at run-time)!!!***

ПАРАДИГМА MANAGER/WORKER

- ***ЦЕЛ*** – да осигури разпределението на задачите между процесите ***в хода на изпълнението на програмата***
  - Процесът ***manager:***

1. Проследява разпределените и неразпределените данни
2. Отговоря за разпределението на задачите между процесите работници (workers)

3. *За един процес – една задача –* ***добър баланс***


Manager
Разпределена задача
Изпълнена задача
Разпределена задача
Изпълнена задача
Разпределена задача
Изпълнена задача
Worker
Worker
Worker

Моделът manager/worker vs *SPMD* (Single Program Multiple Data) "spim-dee" стил

- Всеки процес изпълнява едни и същи функции
- моделът manager/worker се различава радикално от стила SPMD
- Процеса manager е # от процесите worker
- Разцепване на управляващия поток
- Manager – идентифициране на документите
- Workers – четене на каталога

©BOROVSKA

Процес manager

- Идентифицира специфицираните от потребителя *n* текстови документи в каталога
- Получава от worker 0 броя на елементите ***k*** на вектора на документа
- ***allocate матрица s [nxk] за векторите, които получава от workers***
- Инициализира променливите ***d*** (не са разпределени документи за обработка) и ***t*** (никой процес worker не е терминиран)
- Влиза в ***цикъл*** – докато не терминира всички процеси workers

©BOROVSKA

Процес manager

- В рамките на цикъла получава съобщение от процес worker
- Съобщението съдържа профилиращия вектор на документа
- Процес manager съхранява вектора на съответ-ната позиция в матрицата на векторите *s*
- В противен случай – процесът worker просто индицира, че е готов да се заеме с обработката на документ – ***manager му изпраща file ID, записва в масив а кой документ е разпределен и инкрементира d***

Процес manager

- Ако всички документи вече са обработени – изпраща ***съобщение за край на обработката (termination message)*** на процеса worker и ***инкрементира броя на терминираните процеси (termination count)***
- Повтаря цикъла докато терминира всички процеси worker
- ***При излизане от цикъла manager записва профилиращите вектори на документите в матрица s във файл***

MANAGER (псевдокод)

Локални променливи

- ***a*** – масив с разпределените документи между процесите
- ***d*** – разпределените за обработка документи
- ***j*** – worker ID, подал заявка за документ
- ***k*** – дължина на вектора на документа
- ***n*** – общ брой на документите
- ***p*** - общ брой на процесите: (p-1) са workers
- ***s*** – масив, съдържащ векторите на документите
- ***t*** – брой на терминираните процеси worker
- ***v*** – индивидуален вектор на всеки документ

©BOROVSKA

Идентифицира n документа в каталог, специфициран от потребителя

Получава размера на каталога k от worker 0

Разпределя матрица s[nxk] за векторите на документите

d←0

t←0

repeat

получава съобщение от worker j

if съобщението съдържа вектор на документ

s[a[j]] ← v

else

{съобщението съдържа първа заявка за работа}

endif

©BOROVSKA

```
    endif
    if d<n then
        send името на документа d на worker j
        a[j] ← d
        d ← d + 1
    else
        send termination message на worker j
        t ← t + 1
    endif
until t = p – 1
запис на s в изходен файл
```

Функция MPI_Abort

- Manager разпределя матрицата s[nxk] за векторите на документите
- If memory allocation fails → терминираме изпълнението на MPI програмата
- Функция MPI_Abort **header:**

Int MPI_Abort (MPI_Comm comm, int error_code)

- Прекратява изпълнението на всички процеси в рамките на комуникатора (comm)
- Връща стойността на error_code

Процес worker

- Всеки процес worker трябва да разполага с копие на каталога
- Възможно решение – всеки worker да отвори файла с каталога и да прочете съдържанието
- Друга опция – един worker да отвори файла с каталога и да прочете съдържанието, след което да изпрати каталога на останалите workers (**broadcast**) – **предпочита се ако пропускателната способност за broadcast на SAN е по-голяма на тази на file server**

©BOROVSKA

WORKER (псевдокод)

Worker

Локални променливи

f – име на файла

k – размер на каталога

v – вектор на документа

send първата заявка за работа към manager

if worker 0 then

прочита каталога от файла

endif

©BOROVSKA

```
broadcast каталога на всички останали
  workers
конструира hash table от каталога
if worker 0 then
    send размера k на каталога на manager
endif
repeat
    receive file ID от manager
    if f индицира терминиране
        exit loop
    else …
```

```
    else
        чете документа от файла f
        генерира вектора v на документа
        send вектора v на manager
    endif
forever
```

- Worker известява manager в момента, в който стане готов за работа (фактически не е готов, докато не е изграден каталога)

- **Ранна заявка за работа** (съвместяване на t за send и t за получаване на първия file ID от manager с t за съставяне на каталога
  - Worker 0 чете каталога
- **Всички процеси worker (без manager) участват в broadcast**
  - **Всеки worker съставя хешираща таблица** от елементите на каталога – осигурява се const t за достъп до позициите на каталога и **се ускорява профилирането на документите**

©BOROVSKA

- Процес worker влиза в цикъл **repeat … forever**
- worker получава съобщение от manager
- Ако съобщението съдържа file ID, worker чете файла, генерира вектора на документа, изпраща го на manager, и преминава към следващата итерация на цикъла
- Ако worker получи съобщение terminate от manager, това означава, че няма необработени документи и worker престава да съществува
- **Решение, кой от процесите да бъде manager – за простота се избира процес с ранг 0 в MPI_COMM_WORLD**

КАТАЛОГ INFO

ФАЙЛ С РЕЗУЛТАТИТЕ

Manager

ФАЙЛ С ДОКУМЕНТ

Worker 2

Worker 3

ФАЙЛ С ДОКУМЕНТ

ФАЙЛ С ДОКУМЕНТ

Worker 0

Worker 1

ФАЙЛ С ДОКУМЕНТ

ФАЙЛ С КАТАЛОГА

Комуникатор на процесите worker

- Създаваме нов комуникатор, обхващащ всички процеси worker без процеса manager
- Разделяне на комуникатор на един или повече нови комуникатори с MPI_Comm_Split
- За да изключим manager, той предава константата MPI_UNDEFINED като стойност на split_key; върната стойност на new_comm ще бъде MPI_COMM_NULL

Комуникатор на процесите worker

```
int id;
MPI_Comm worker_comm;
……..
If (!id) /* manager */
   MPI_Comm_split (MPI_COMM_WORLD,
   MPI_UNDEFINED, id, &worker_comm) ;
Else /* worker */
   MPI_Comm_split (MPI_COMM_WORLD, 0, id,
   &worker_comm) ;
```

Group routines

- MPI_Group_difference създава група от разликата между две групи
- MPI_Group_incl създава група от изброени членове на съществуваща група
- MPI_Group_excl създава група, изключваща изброени членове на съществуваща група
- MPI_Group_range_incl създава група (първи ранг, стъпка, последен ранг)
- MPI_Group_range_excl създава група чрез изтриване (първи ранг, стъпка, последен ранг)
- MPI_Group_free маркира група за deallocation

©BOROVSKA

Communicator routines

- **MPI_Comm_size** връща броя на процесите в групата на комуникатора
- **MPI_Comm_rank** връща броя на извикващите процеси в групата на комуникатора
- **MPI_Comm_compare** сравнява два комуникатора
- **MPI_Comm_dup** създава копие на комуникатора
- **MPI_Comm_create** създава нов комуникатор за дадена група
- **MPI_Comm_split** разцепва комуникатора на множество неприпокриващи се комуникатори
- **MPI_Comm_free** маркира комуникатор за deallocation

Режими на комуникация

Комуникация Режим	Blocking Routines	Non-Blocking Routines
Синхронни	MPI_SSEND	MPI_ISSEND
Ready	MPI_RSEND	MPI_IRSEND
Буферирани	MPI_BSEND	MPI_IBSEND
Стандартни	MPI_SEND	MPI_ISEND
	MPI_RECV	MPI_IRECV

©BOROVSKA

Режими на комуникация

- **Buffer-mode**: операцията send може да бъде стартирана и завършена независимо от това дали съответната операция receive е инициирана.
- **Synchronous-mode**: = buffer mode, но операцията send ще завърши успешно, само ако съответната операция receive е инициирана, и операцията receive е стартирана да получи съобщението, изпратено от синхронен send.
- **Ready-mode send** може да бъде стартирана само ако съответната операция receive вече е инициирана.
- **Standard communication mode:** MPI решава дали изходящите съобщения ще бъдат буферирани. Операция send в стандартен режим може да бъде стартирана независимо от това дали съответната операция receive е инициирана.

Режими на комуникация

	Предимства	Недостатъци
Synchronous	най-безопасен, портабилен; редът на SEND/RECV не е критичен, размерът на буфера не е проблемен	значителен overhead за синхронизация
Ready	най-малък общ overhead SEND/RECV handshake не се изисква	RECV трябва да предхожда SEND
Buffered	отделя SEND от RECV няма sync overhead при SEND редът на SEND/RECV не е от значение програмистът може да управлява размера на буфера	допълнителен system overhead за копиране в буфера
Standard	добър за много случаи	програмата може да не е подходяща

©BOROVSKA

БЛОКИРАЩИ КОМУНИКАЦИИ

- **MPI_Send (start, count, datatype, dest, tag, comm)**
- **MPI_Recv (start, count, datatype, source, tag, comm, status)**
  - **Start:** начален адрес на буфера
  - **Count:** (max) брой на получените елементи
  - **Datatype:** описател на типа на получените данни; може да опише произволна (прекъсната) област данни в паметта
  - **Source:** ранг на процеса в групата на комуникатора; може да бъде MPI_ANY_SOURCE
  - **Tag:** целочислен идентификатор на съобщението; може да бъде MPI_ANY_TAG

©BOROVSKA

БЛОКИРАЩИ КОМУНИКАЦИИ

- **Комуникатор:**
  - специфицира наредена група комуникиращи процеси
  - специфицира домейн на комуникацията. Съобщение, изпратено с един комуникатор, може да бъде получено само със "същия" комуникатор
- **Status:** осигурява info за завършена комуникация

©BOROVSKA

MPI message

- Message = data + envelope
- MPI_Send (startbuf, count, datatype,

dest, tag, comm)

©BOROVSKA

НЕБЛОКИРАЩИ КОМУНИКАЦИИ

- Работата на процеса manager вкл. 3 фази

1. Намира текстовите файлове в каталожната структура, специфицирана от потребителя, получава размера на каталога от worker 0 и разпределя двумерния масив за съхраняване на профилиращите вектори на документите
2. Разпределя документите за обработка от процесите workers и събира профилиращите вектори
3. Записва пълното множество от профилиращи вектори във файл

©BOROVSKA

Manager – фаза 1

- ? Възможностите за припокриване на двете дейности – търсене в каталожната структура и получаване на съобщение от worker 0
- Съобщения от точка до точка ***(poit-to-point message passing):* MPI_Send и MPI_Recv *(blocking operations)→ограничават производителността на паралелната програма!!!***
- Функцията **MPI_Send** блокира процеса докато съобщението не бъде копирано в системния буфер или не бъде изпратено
- Функцията **MPI_Recv** блокира процеса докато съобщението не бъде получено в буфера, специфициран от потребителя

©BOROVSKA

- Изпълнението на receive преди да пристигне съобщението спестява време, тъй като системата съхранява операцията за копиране чрез прехвърляне на *съдържанието на входящото съобщение директно в буфера-дестинация*, а не във временен системен буфер
- Извикванията на неблокиращите функции ***MPI_Isend и MPI_Irecv просто инициират съответните операции за комуникация***

(I – initiate)

- потребителският процес не може да осъществи достъп до буфера докато комуникацията не завърши явно с изивкване на MPI_Wait

Комуникация на manager

- **Функция MPI_Irecv header:**

Int MPI_Irecv (void *buffer, int count, MPI_Datatype dtype, int src, int tag, MPI_Comm comm, MPI_Request *handle)

- MPI_Irecv само инициира получаването → не може да се осъществи достъп до буфера, докато не се върне съответното извикване на MPI_Wait
- Връщането се осъществява чрез последния параметър handler (указател) към обекта на MPI_Request, определящ инициираната комуникация

- **Функция MPI_Wait header**

Int MPI_Wait (MPI_Request *handle, MPI_Status *status)

- Функцията блокира докато операцията, свързана с указателя handle завърши
- В случая на операция ***send*** на буфера могат да бъдат присвоени нови стойности
- В случая на операция ***receive*** може да се осъществи достъп до буфера и status указва обекта MPI_Status, съдържащ info за полученото съобщение

Комуникациите на workers

- Worker трябва да получава имена на файлове (пълни каталожни пътища) от manager
- Няма предварителна информация за каталожните пътища
- Worker трябва да може да определя дължината на входящите съобщения, преди да ги прочете
- За тези цели се използват следните три MPI функции: *MPI_Isend, MPI_Probe, MPI_Get_count*

©BOROVSKA

Функция MPI_Isend

Int MPI_Isend (void *buffer, int cnt, MPI_Datatype dtype, int dest, int tag, MPI_comm comm, MPI_request *handle)

- Инициира неблокираща операция send
- Последният параметър е изходен – handle за прозрачен MPI_Request обект, създаден от run-time system
- Той определя тази заявка за комуникация
- Буферът не може да се използва отново до връщането на MPI_wait

©BOROVSKA

Функция MPI_Probe

Int MPI_Probe (int src, int tag, MPI_Comm comm, MPI_Status *status)

src – ранг на подателя на съобщението

tag – на входящото съобщение

comm – комуникатор

status – указател към MPI_Status обект

- Функцията блокира, докато съобщение от съответния подател и таг е налично за приемане
- Връща чрез указателя status info за подателя, тага и дължината на съобщението
- Фактически не получава съобщението

Функция MPI_Probe

- Като се подава MPI_ANY_SORUCE като src аргумент, може да се проверява за съобщение от всеки процес
- MPI_ANY_TAG → проверява се за съобщение от процеса, определен с аргумента src
  - MPI_ANY_SORUCE и MPI_ANY_TAG → проверява се за съответствие на всяко съобщение
- Спецификациите на src & tag трябва да бъдат по възможност най-тясно задавани → избягване на съобщения, получавани по грешка

©BOROVSKA

Функция MPI_Get_count

Int MPI_Get_count (MPI_Status *status, MPI_Datatype dtype, int *cnt)

- Предава указател status към MPI_Status обект
- dtype – типа на елементите на съобщението
- cnt – указател към int, чрез него се връща броят на елементите в съобщението

ПОДОБРЕНИЯ

1. **РАЗПРЕДЕЛЯНЕ НА ГРУПИ ОТ ДОКУМЕНТИ**

- Разпределянето на документите между процесите в хода на изпълнението на програмата води до добър баланс на паралелния изчислителен товар, но води до допълнителни разходи за интерпроцесорна комуникация
- Предварително разпределяне → дисбаланс
- Междинно решение – парадигма manager/worker при която manager разпределя в даден момент по k задачи на всеки worker

2. КОНВЕЙЕРИЗАЦИЯ (PIPELINING)

- Определихме идентифицирането на текстовите файлове и записа на резултатния файл като последователни задачи - manager
- Никой документ не може да бъде обработен преди manager да ги идентифицира
- Ако времето за изпълнението на тези задачи не е пренебрежимо малко, то програмата няма да бъде ефективна при платформа с голям брой процесори (следствие от закона на Амдал)

- *Разделяме задачата за идентифицирането на документите на елементарни подзадачи*
- Възможност за ***конвейеризация*** на обработката на документите
- ***Идентифицирането на документ i+1 може да се съвмести с четенето на документ i и генерирането на вектора на i-1 документ***
- ***Конвейерното изпълнение може драстично да намали времето за изпълнение на паралелните програми с функционален паралелизъм***
- ***Конвейеризираният вариант с 1 процес за четене, два процеса worker и 1 процес за запис осигурява по-висока производителност от неконвейеризирания вариант с 1 manager и 3 workers***

Идентифи-
циране на
документ i+2

Четене на
документ i+2

Генериране
на вектор за
документ i+2

Запис на
вектора на
документ 1

Идентифи-
циране на
документ i

Четене на
документ i

Генериране
на вектор за
документ i

Запис на
вектора на
документ i

Идентифи-
циране на
документ i-1

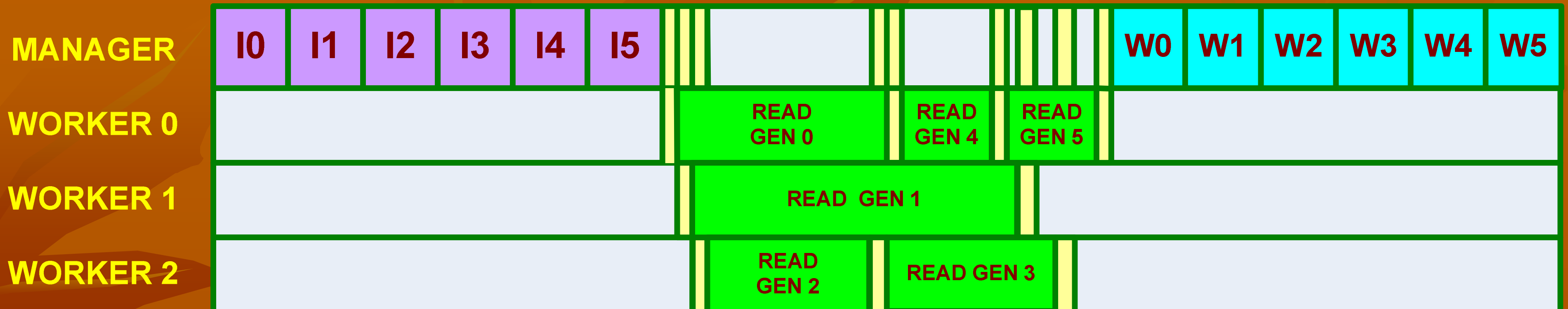
Четене на
документ i-1

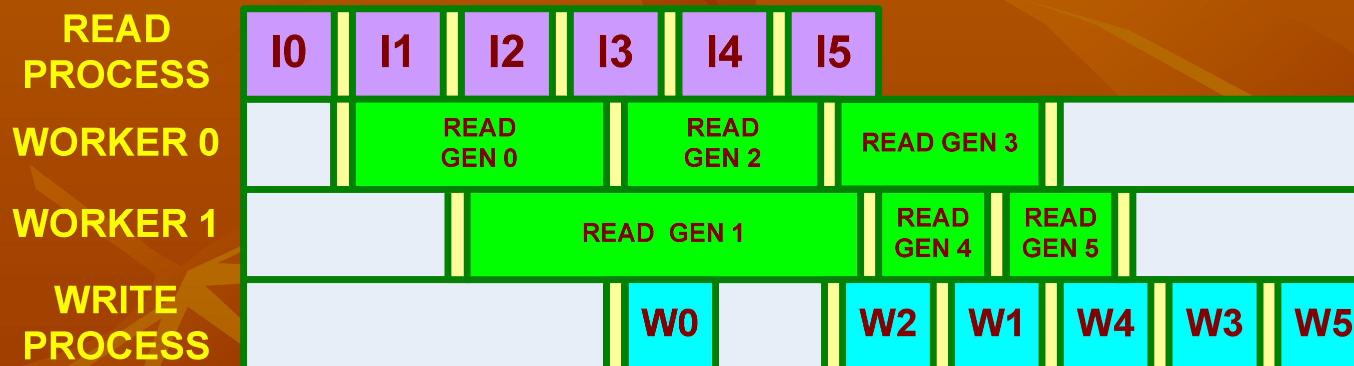
Генериране
на вектор за
документ i-1

Запис на
вектора на
документ i-1

2. КОНВЕЙЕРИЗАЦИЯ (PIPELINING)

©BOROVSKA





2. КОНВЕЙЕРИЗАЦИЯ (PIPELINING)

©BOROVSKA

КРАЙ