ПРОГРАМИРАНЕ С ОБМЕН НА СЪОБЩЕНИЯ

- *Стандартът MPI (Message Passing Interface) е най-популярната спецификация за обмен на съобщения, поддържаща паралелното програмиране*
- Виртуално всеки комерсиален паралелен компютър поддържа MPI
- Библиотеките са свободно достъпни за ползване на произволен клъстер

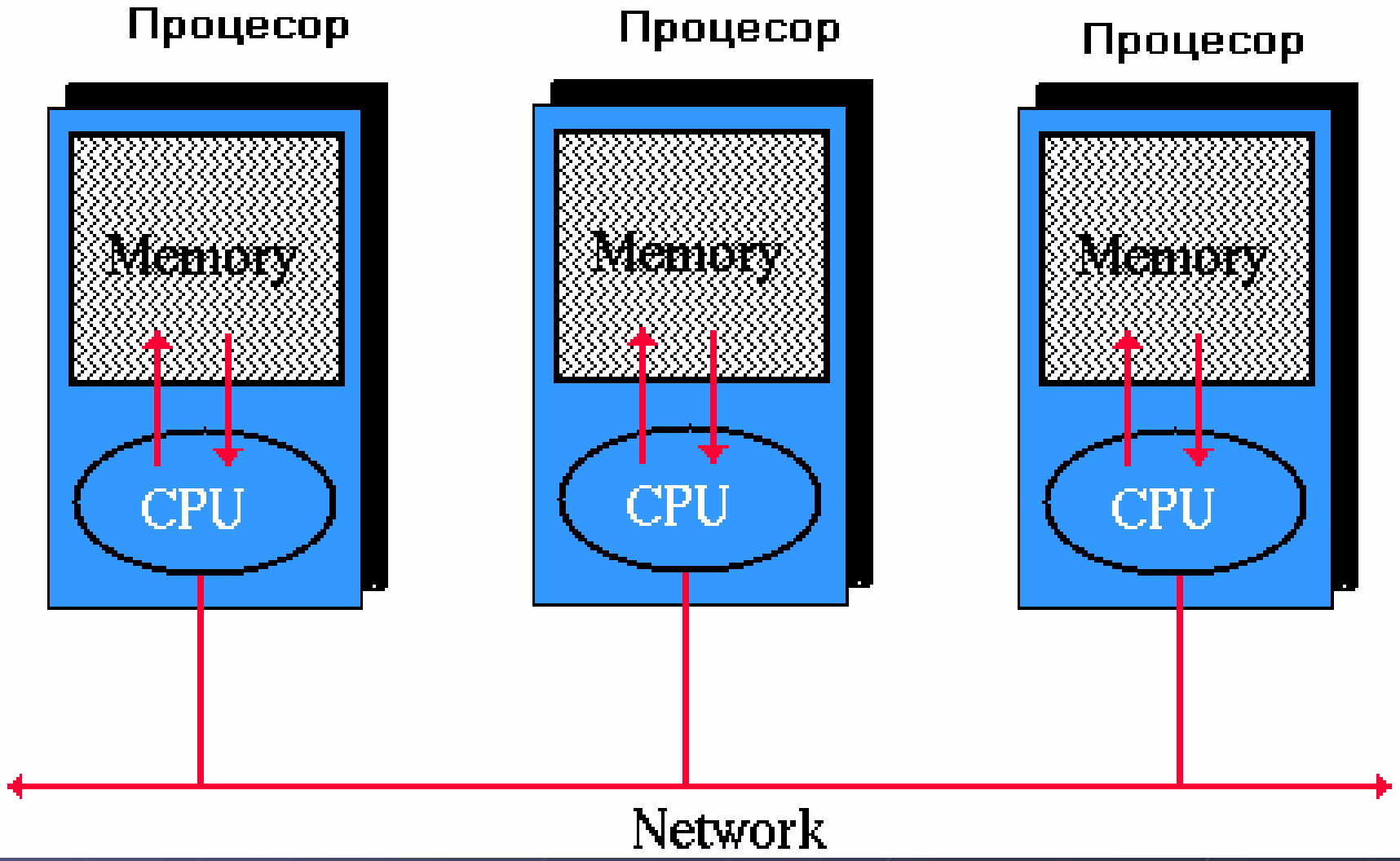
Функции :

- ***MPI_Init*** – инициализиране на MPI
- ***MPI_Comm_rank*** – определяне на ID на процес
- ***MPI_Comm_size*** – определяне на броя на процесите
- ***MPI_Reduce*** – операция редукция
- ***MPI_Finalize*** – shutdown MPI
- ***MPI_Barrier*** – бариерна синхронизация
- ***MPI_Wtime*** – определяне на времето
- ***MPI_Wtick*** – точност на таймера

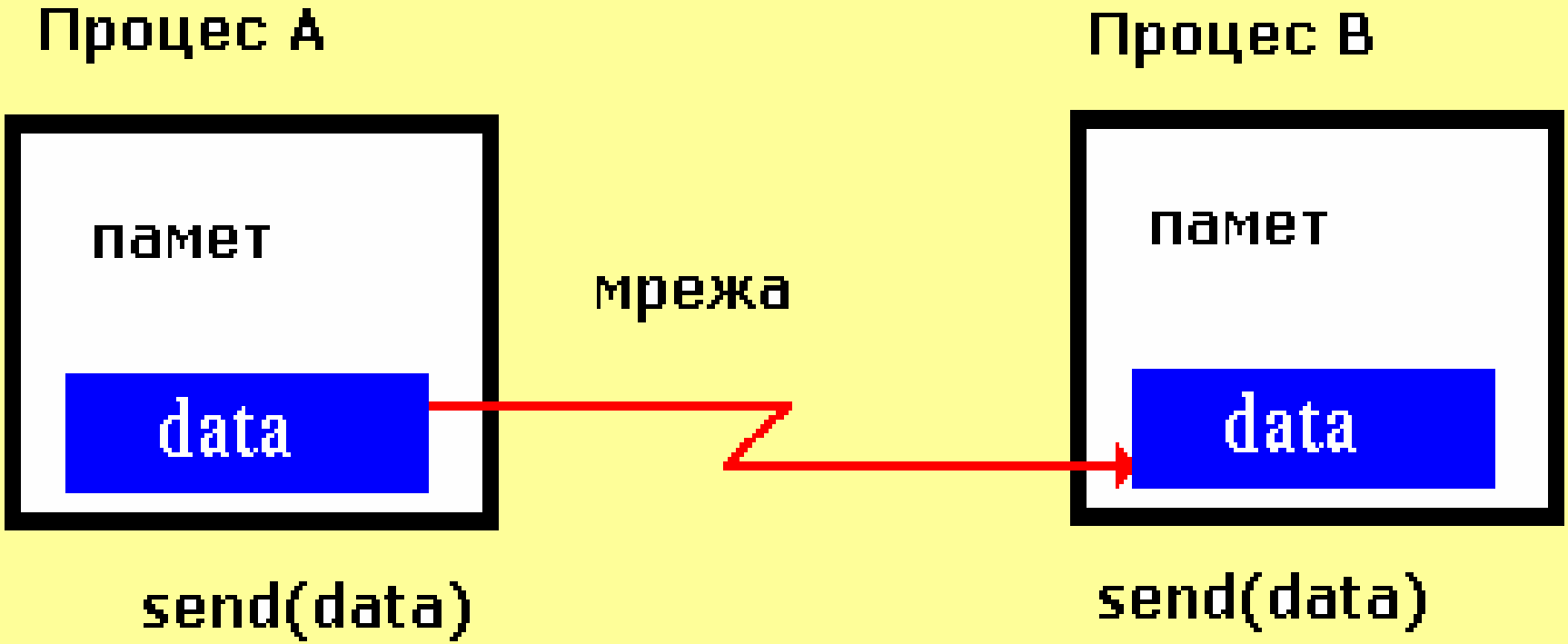
The message-passing model

- Множество процесори, всеки с локална памет
- Системната комуникационна мрежа поддържа обмена на съобщения между всяка двойка процесори
- Всяка задача от графа на задачите става процес
- Всеки процес може да комуникира с всеки друг процес

Система на разпределената памет



Предаване на съобщенията



- При стартирането на програмата се определя броят на паралелните процеси
- В общия случай този брой остава *постоянен* в хода на изпълнение на програмата
- Всеки процес алтернативно извършва изчисления над локалните си променливи и комуникира с останалите процеси или В/И устройства
- *Процесите използват съобщенията както за комуникация, така и за синхронизация*

- *Когато съобщението съдържа данни →*
  за комуникация
- *Когато съобщението не съдържа данни*
  – за синхронизация

Предимства на модела с обмен на съобщения

- *Ефективен за широк спектър MIMD архитектури*
- *Естествена среда за мултикомпютърни платформи, които не поддържат глобално адресно пространство*

- *Възможно е да се изпълняват програми с обмен на съобщения на **мултипроцесорни платформи** като общите променливи се използват като буфери на съобщенията*
- *Стимулира синтеза на алгоритми, които максимално използват локални изчисления и минимизират комуникациите*
- *Осигурява висока честота на попаденията в кеша – ефективно управление на управлението на паметта*

- *Настройката на програми с обмен на съобщения е по-лесна от тази на програми с общи променливи*
  - *По-лесно се изпълняват детерминистични програми*
- *Недетерминистичното изпълнение затруднява настройката – при различните изпълнения на паралелната програма различните процеси заемат едни и същи ресурси в различен ред*

- *Първата версия на библиотеката за обмен на съобщения PVM (Parallel Virtual Machine) е създадена в Oak Ridge National Laboratory*
- *PVM – изпълнение на паралелни програми на хетерогенни колекции от паралелни и последователни машини*
- *1993 г. – публикувана версия 3*

Geist, Al, Adam Bequelin, Jack Dongarra, et all., "PVM: Parallel Virtual Machine: A User's Guide and Tutorial for Networked Parallel Computing, Cambridge, MA: The MIT Press, 1994.

The MPI Forum

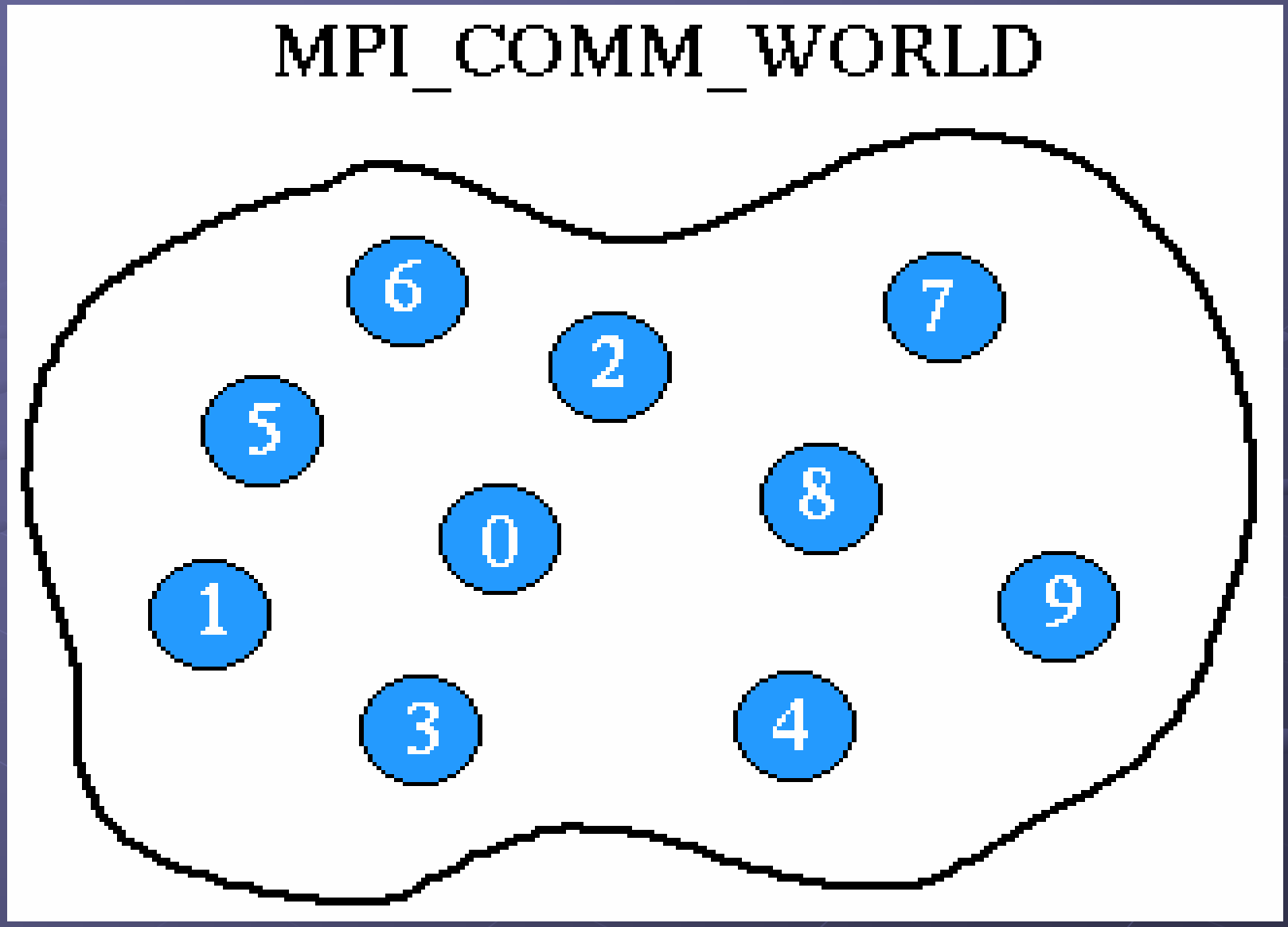
- Спонсориран от Center for Research on Parallel Computing – 1992 г. – представители на производителите, университети, правителствени лаборатории, индустрията (общо 40 организации от САЩ и Европа)
  - Version 1.0 – 1994 г.
  - Официално стандартът е приет през 1997 г.
  - Преносимост – при # платформи → # производителност

Функция MPI_Init

- *Първото извикване на MPI функция от всеки MPI процес*
- *Инициализация на системата за да се осигури възможност за извикване на MPI функциите*
- *Не е задължително да бъде първия изпълним оператор, нито да се съдържа във функцията main*
- *Всички идентификатори започват с префикса MPI_ :* **MPI_Init (&argc, &argv)**

Функции MPI_Comm_rank и MPI_Comm_size

- *След инициализацията, всеки активен процес става член на комуникатор, наречен* **MPI_COMM_WORLD** *(default communicator)*
- *Комуникаторът представлява прозрачен обект, осигуряващ среда за обмен на съобщения между процесите*
- *Собствени комуникатори – процесите се разделят на независими комуникиращи групи*



- *В рамките на комуникатора процесите имат строго определен ред*
- *Позицията на процеса в този ред се определя от неговия ранг*
- *В комуникатор с p процеса, всеки процес има уникален ранг (ID номер), вариращ между 0 и p-1*
- *Даден процес може да използва ранга си за да определи за коя част от изчисленията и данните е отговорен*

- *Даден процес може да извика функцията MPI_Comm_rank за да определи своя ранг в рамките на комуникатора*
- *С функцията MPI_Comm_size може да се определи общият брой на процесите в даден комуникатор*

Функция MPI_Finalize

След завършване на всички MPI извиквания, процесът извиква *MPI_Finalize* за да освободи всички системни ресурси (като напр. памет)

Компилиране на MPI програми

Командата е различна за различните системи

% mpicc –o myprog myprog.c

При тази команда системата компилира MPI програмата от файла *myprog.c* и записва изпълнимия код във файла *myprog*

Изпълнение на MPI програми

% mpirun -np 2 myprog

Флагът ***–np*** показва броя на процесите, които трябва да бъдат създадени

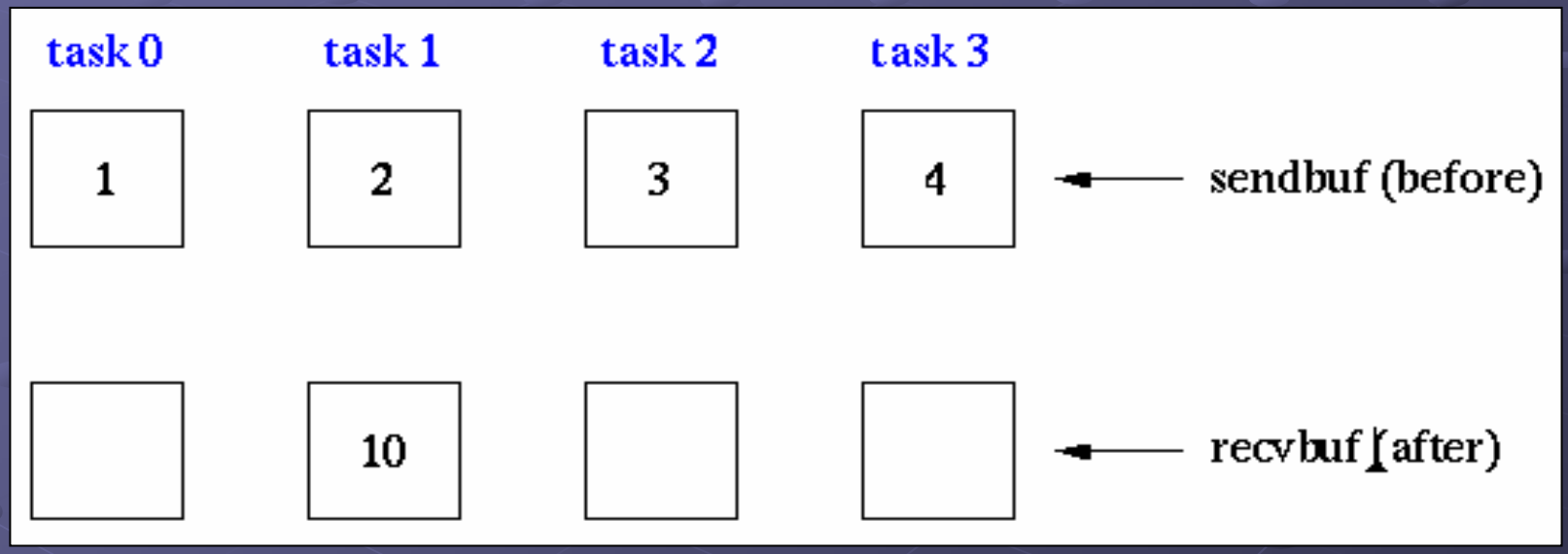
КОЛЕКТИВНА КОМУНИКАЦИЯ

- Колективната комуникация е операция за комуникация, при която група процеси заедно разпределят или събират една или множество стойности.

Функция MPI_Reduce

- Функцията изпълнява една или повече операции редукция върху стойности, предадени от всички процеси в комуникатора

Функция MPI_Reduce



```
int MPI_Reduce  (
    void            *operand,   /* адрес на 1 елемент */
    void            *result,    /* адрес на 1 резултат */
    int             count       /* брой редукции */
    MPI_Datatype    type,       /* тип на елементите */
    MPI_Op          operator,   /* оператор редукция */
    int             root,       /* процес(ранг) ← резултат */
    MPI_Comm        comm)       /* име на комуникатор */
```

Въпреки, че само един процес получава глобалния резултат, всички процеси в комуникатора трябва да извикат функцията MPI_Reduce, за да се включат в колективната комуникация!!!

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

вградени MPI операции за редукция

- MPI_BAND
- MPI_BOR
- MPI_BXOR
- MPI_LAND
- MPI_LOR
- MPI_LXOR
- MPI_MAX
- MPI_MAXLOC
- MPI_MIN
- MPI_MINLOC
- MPI_PROD
- MPI_SUM

BENCHMARKINK PARALLEL PERFORMANCE

ПОЛЗИТЕ ОТ ПАРАЛЕЛНОТО ИЗПЪЛНЕНИЕ

Функциите MPI_Wtime и MPI_Wtick

- За измерване на производителността – време по стенен часовник (секунди) – от стартирането до завършването на програмата
- В общия случай игнорираме времето за инициализация на MPI процесите, конфигурирането на комуникациите между тях и В/И операции на последователните устройства

- **Функцията MPI_Wtime връща броя на секундите, отброени от фиксиран минал момент**
- **Функцията MPI_Wtick връща точността на резултата от MPI_Wtime.**

double MPI_Wtime (void)
double MPI_Wtick (void)

Можем да определим бързодействието при изпълнението на секция от кода, като поставим две извиквания на функцията MPI_Wtime преди и след секцията. Разликата между двете стойности, връщани от функцията, дават изминалите sec.

Функция MPI_Barrier

- Никой от процесите не може да продължи след бариерата преди всички процеси да са стигнали до нея
- Ако тя е преди първото извикване на MPI_Wtime, всички процеси ще се вместят в рамките на измерваната секция приблизително по едно и също време

Прототип на функцията за бариерна синхронизация:

Int MPI_Barrier (MPI_Comm comm)

- Аргументът указва процесите в комуникатора, участващи в бариерната синхронизация
- Прибавяме локална променлива във функцията *main*:

```
double elapsed_time;
```

- Таймерът се стартира след инициализирането на MPI:

MPI_Init (&argc, &argv);

MPI_Barrier (MPI_COMM_WORLD);

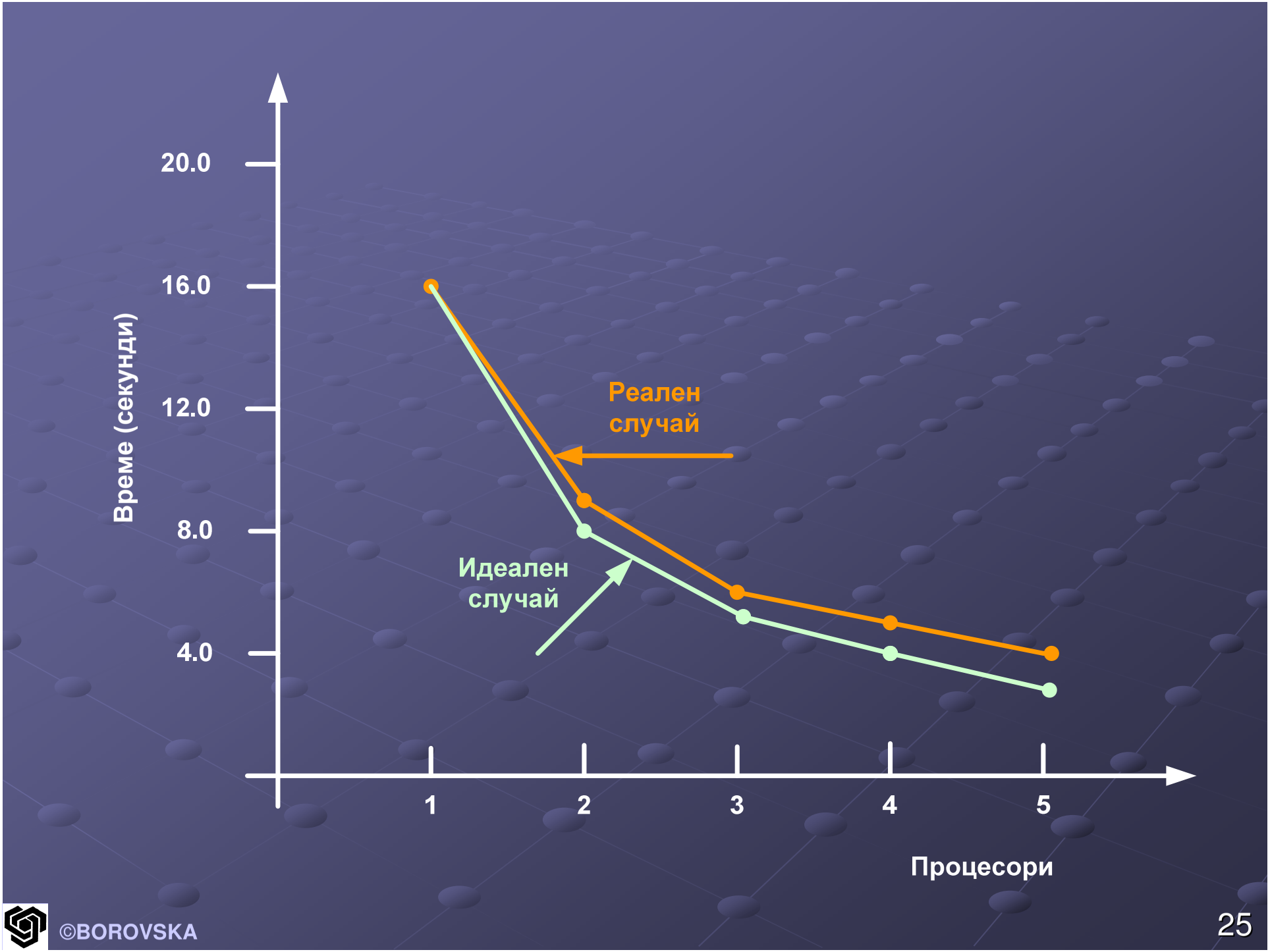
Elapsed_time = - MPIWtime ();

- След извикването на MPI_Reduce можем да спрем таймера:

MPI_Reduce (…..);

Elapsed_time + = MPI_Wtime();

В резултат можем да определим времето за изпълнение на паралелната програма:


20.0
16.0
12.0
8.0
4.0
Време (секунди)
Реален
случай
Идеален
случай
1
2
3
4
5
Процесори

- *Синтеза на паралелни алгоритми с конструиране на граф на задачите естествено води до парадигмата за програмиране с обмен на съобщения*
- *Парадигмата с обмен на съобщения дава възможност на програмиста да управлява използването на паметта и да увеличи локалността*

- Запазването на локалността при обръщенията към паметта е ключов фактор за оптимизиране на производителността както при мултипроцесорите, така и при мултикомпютрите
- Програмите с обмен на съобщения могат да се изпълняват ефективно на широк спектър паралелни компютърни платформи

КРАЙ