# Разработване на паралелни програмни приложения

част 1

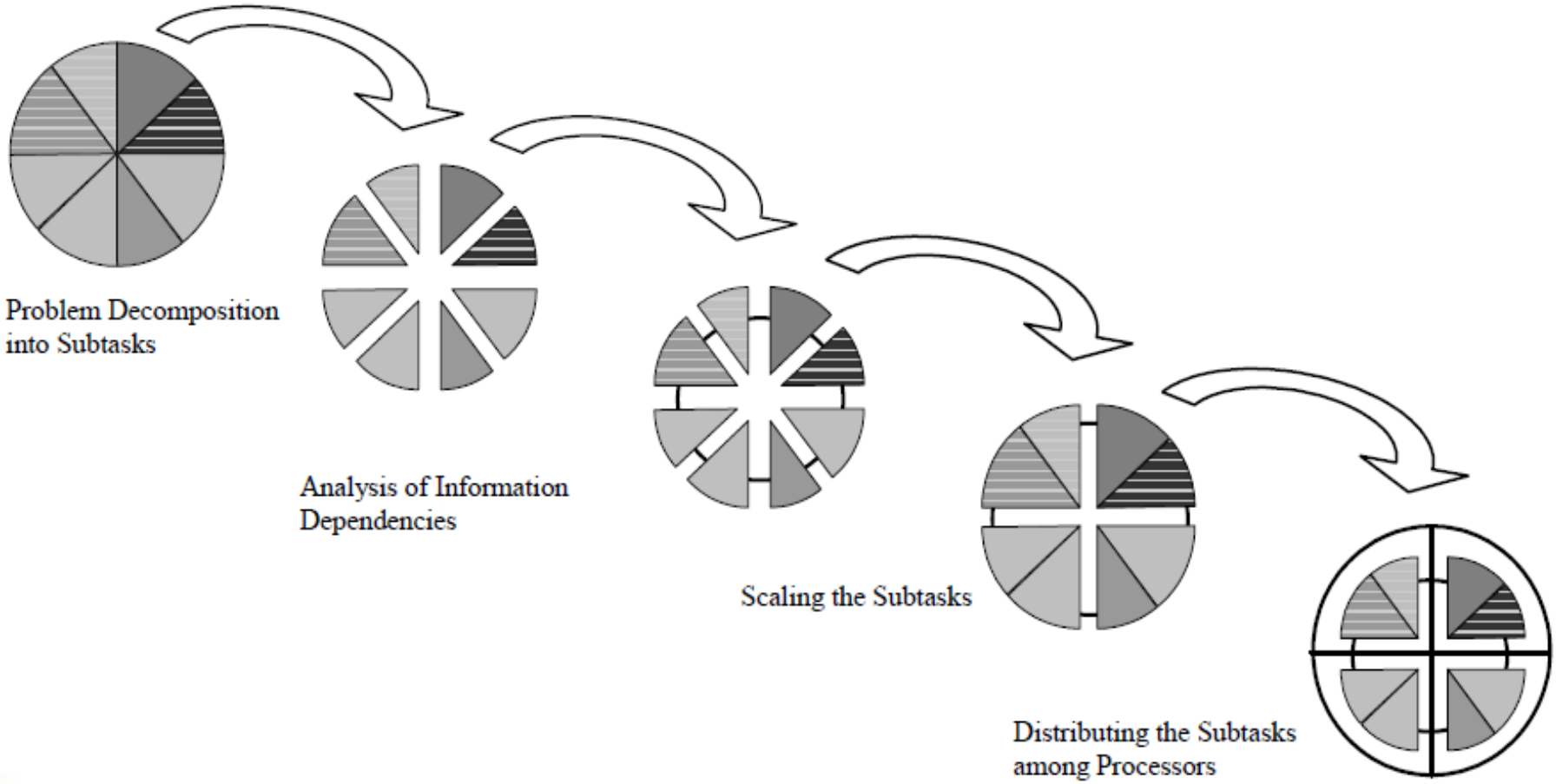
# Синтез на паралелни алгоритми

- Паралелизация на съществуващи последователни алгоритми или модифициране на тези техни части, в които има заложен потенциален паралелизъм
- Синтез на нов паралелен алгоритъм, който може да се адаптира към особеностите на паралелната архитектура
- Синтез на нов паралелен алгоритъм от съществуващ паралелен алгоритъм

# Синтез на паралелни алгоритми

- Анализ на проблема и декомпозиране на подзадачи, които могат да бъдат изчислявани независимо
- Определяне на взаимодействията, необходими между подзадачите в процеса на решаване на първоначално формулирания проблем
  - ако не се изискват комуникации, то проблемът е “embarrassingly parallel”
- Определяне на целева платформа за решаване на проблема и разпределяне на формулирания набор от подзадачи между процесорите в системата
- Най-добрият последователен алгоритъм не е най-добрия паралелен алгоритъм

# Синтез на паралелни алгоритми

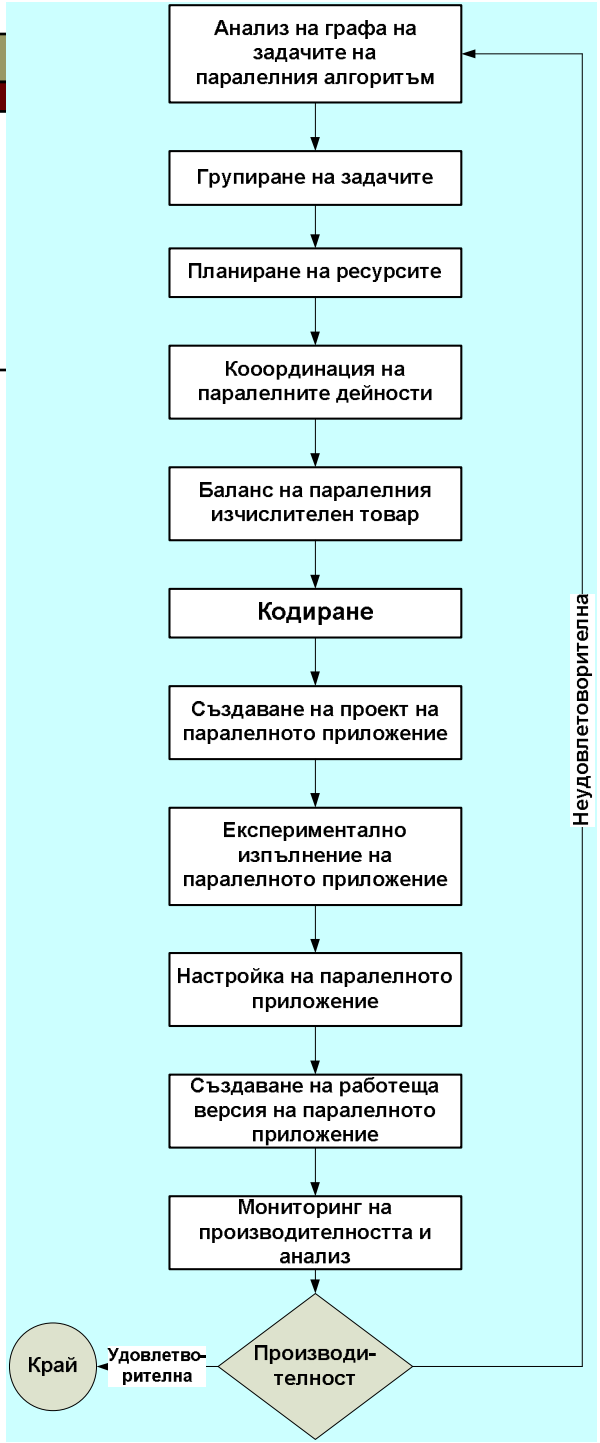


# Синтез на паралелни алгоритми

- За постигане на задоволителна производителност
  - Количеството изчисления за всеки процесор трябва да бъде приблизително еднакъв
    - балансиране на товара (Load Balancing)
  - Разпределението на подзадачи между процесорите да минимизира комуникациите между тях

# Методика за създаване на паралелни приложения

- След преминаване на всички етапи на проектиране и имплементиране на паралелно приложение се оценява ефективността му
  - производителността се оценява на базата на експериментално изследване
- Според резултатите от анализа може да се преразгледат и направят промени по някои (или всички) фази
  - например може да се промени разделянето на подзадачи
    - подзадачите могат да бъдат обобщени (agglomeration) при наличие на малък брой процесори, или обратното
    - подзадачите могат да бъдат прецизирани и броят им увеличен (scaling на разработения алгоритъм)



# Създаване на паралелни приложения

- Паралелно програмиране
  - проектиране, имплементиране и настройване на паралелни програми, изпълнявани на паралелни компютърни системи
- Преходът от последователен изчислителен процес към паралелни изчисления в паралелна компютърна среда въвежда особености при проектирането на паралелни алгоритми и създаването на паралелни програми
  - При проектирането на последователни програми софтуера се разглежда като съвкупност от задачи, които се изпълняват ***последователно*** във времето
  - При паралелното програмиране задачите могат да бъдат разделени на подзадачи, които да се изпълняват ***едновременно***

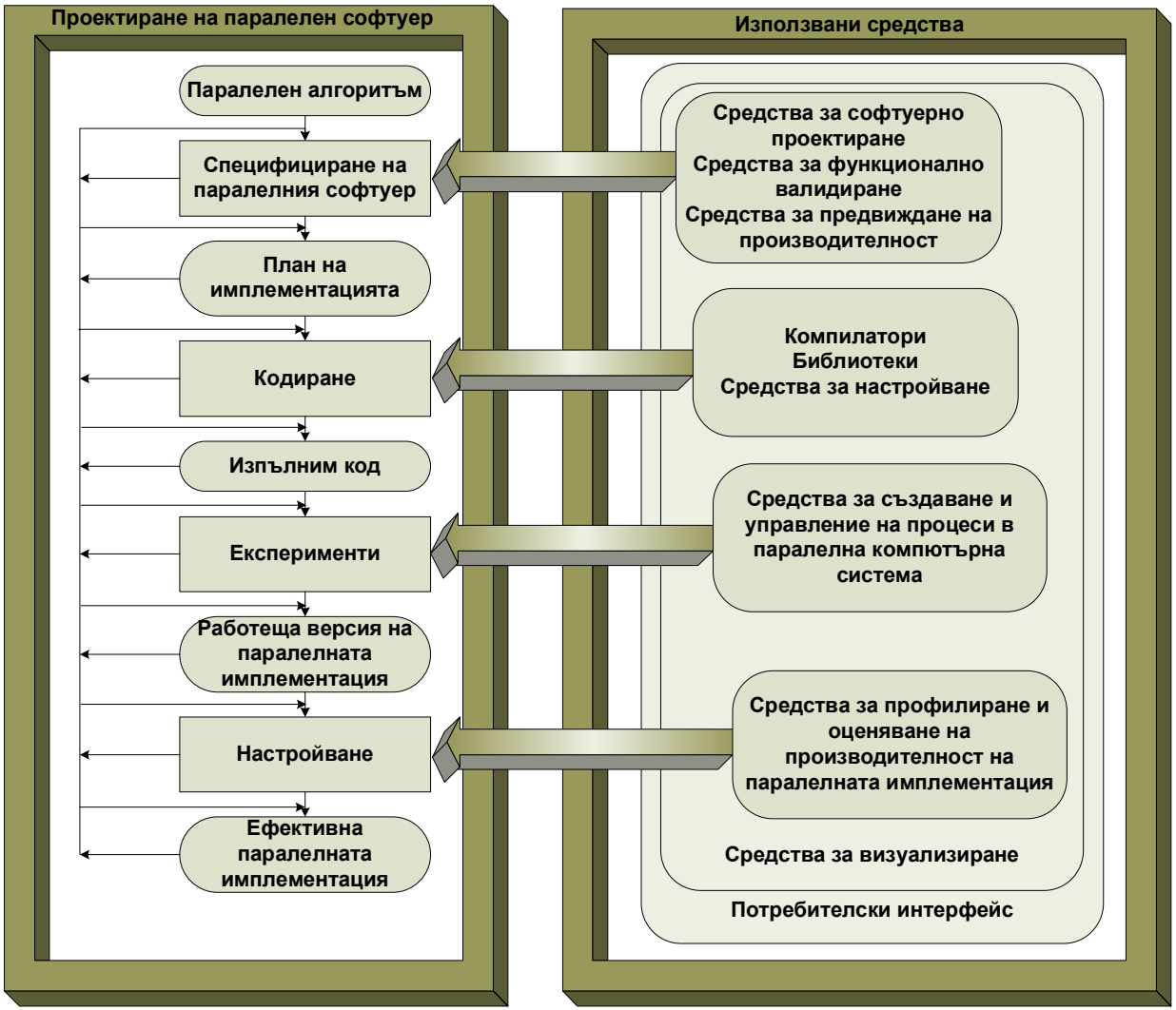
# Създаване на паралелни приложения

- При паралелно изпълнявани задачи
  - не винаги е предварително известен редът на изпълнение и местоположението на задачите и подзадачите във възлите на паралелната компютърна система
    - множество задачи стартират изпълнението си едновременно на някой от процесорите без да е сигурно коя задача ще завърши изпълнението си първа, в какъв ред ще завършат изпълнението си задачите и на кой процесор ще бъде изпълнена всяка от задачите
    - допълнително паралелно изпълнявана задача може да бъде разделена на подзадачи, които също да се изпълняват паралелно
- Паралелни програми, които работят коректно върху еднопроцесорна компютърна система може и по-често няма да работят правилно върху паралелна компютърна система

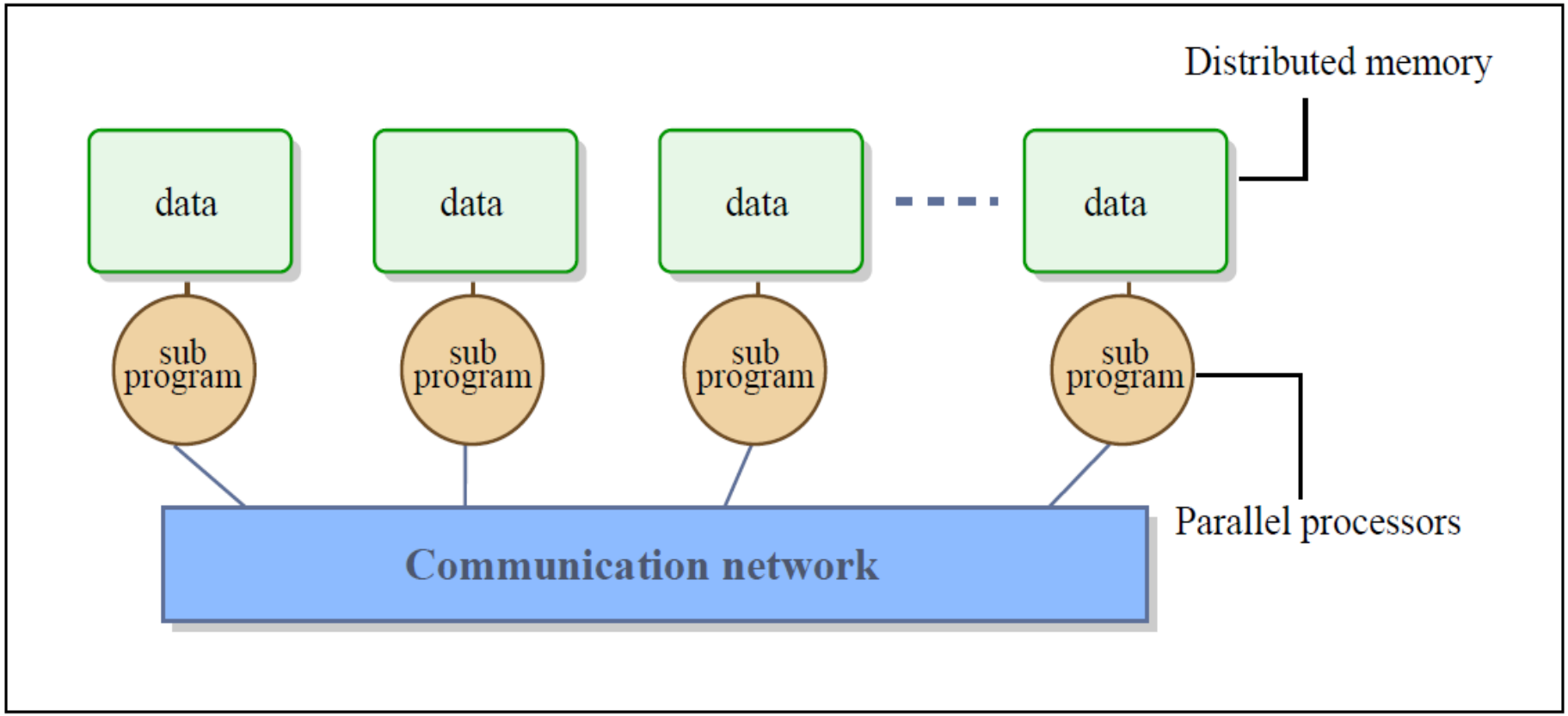
# Създаване на паралелни приложения

- Необходимост от паралелни изчисления възниква при необходимост от
  - *ускоряване на изпълнението*
    - проблем с даден размер да се решава N пъти по-бързо от N процесора
  - *мащабиране*
    - при N-кратно увеличаване на проблема времето за решаването му от N процесора да остава същото
    - т.е. запазване на ускорението при увеличаване на размера на проблема и размера на паралелната машина

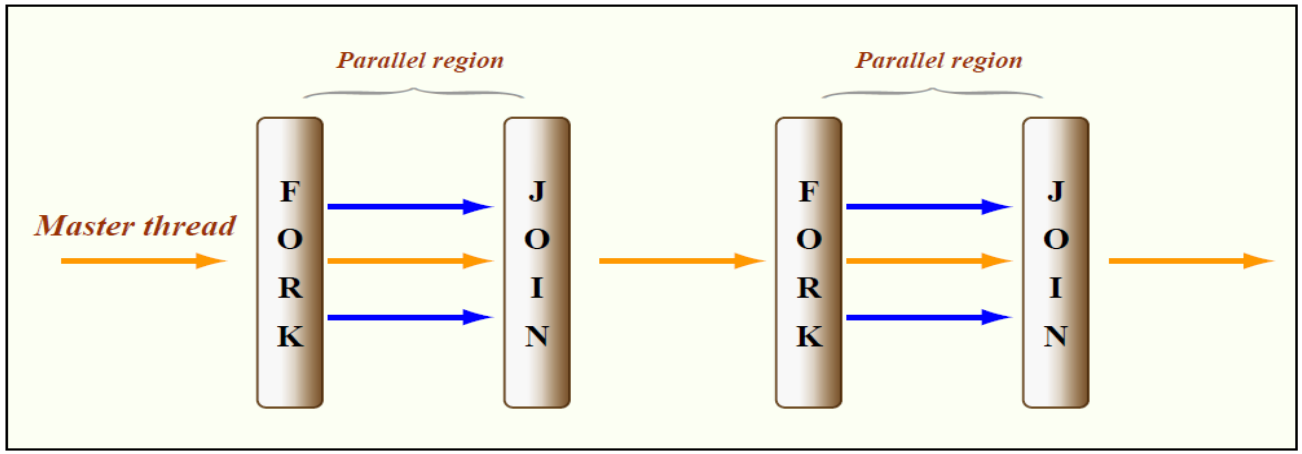
# Създаване на паралелни приложения

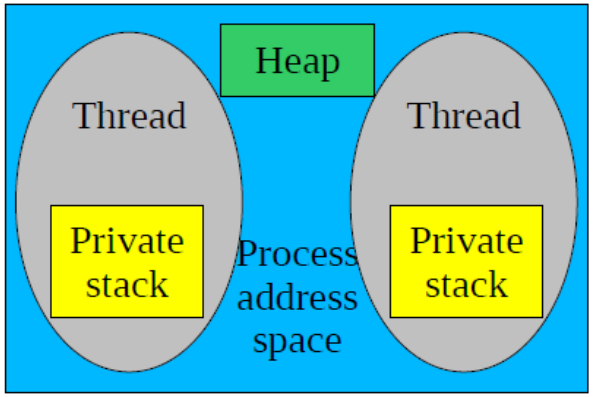


# Плосък модел (MPI)

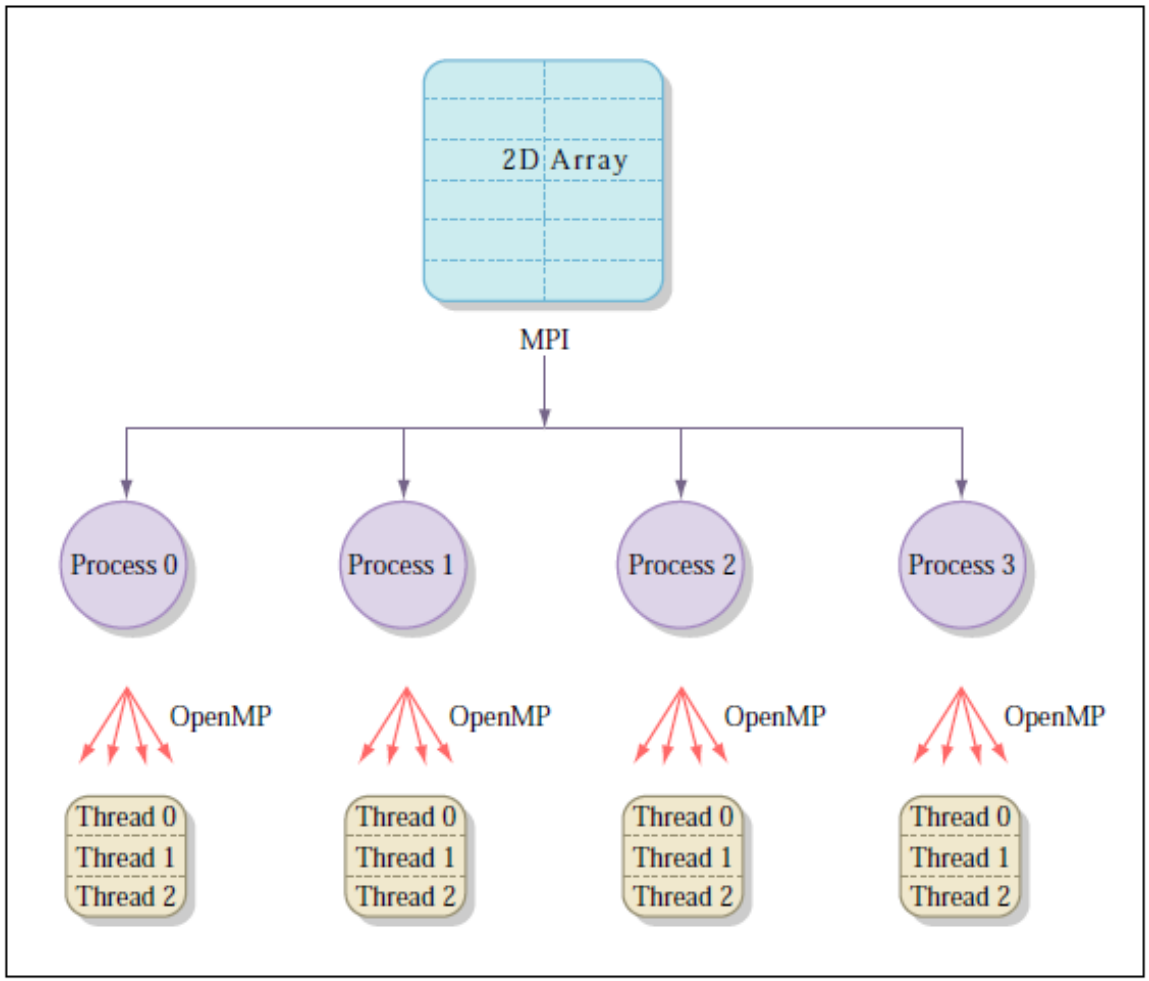


# Многонишков модел (OpenMP)



# Хибриден модел (MPI+OpenMP)



# Примерни паралелни приложения

- Паралелни алгоритми за умножение на матрици
  - Паралелен алгоритъм на Кенън за умножение на матрици
- Паралелни компютърни симулации
  - Паралелно изинг моделиране на прост магнит на основата на алгоритъма Метрополис
  - Паралелна симулация на кръгов трафик по метода Монте Карло
- Паралелно сортиране
  - Паралелно сортиране с контейнери
  - Паралелно пирамидално сортиране
- Паралелна обработка на графи
  - Паралелен алгоритъм на Прим за минимално скелетно дърво
  - Паралелен алгоритъм на Дийкстра за най-кратък път в граф

# Умножение на матрици

- *Задача*
  - да се уможат две матрици като се използва декомпозиция по блокове с алгоритъм на Кенън
- *Целева паралелна компютърна платформа*
  - клъстер от работни станции
- *Програмен модел*
  - плосък (flat)
- *Имплементация*
  - MPI

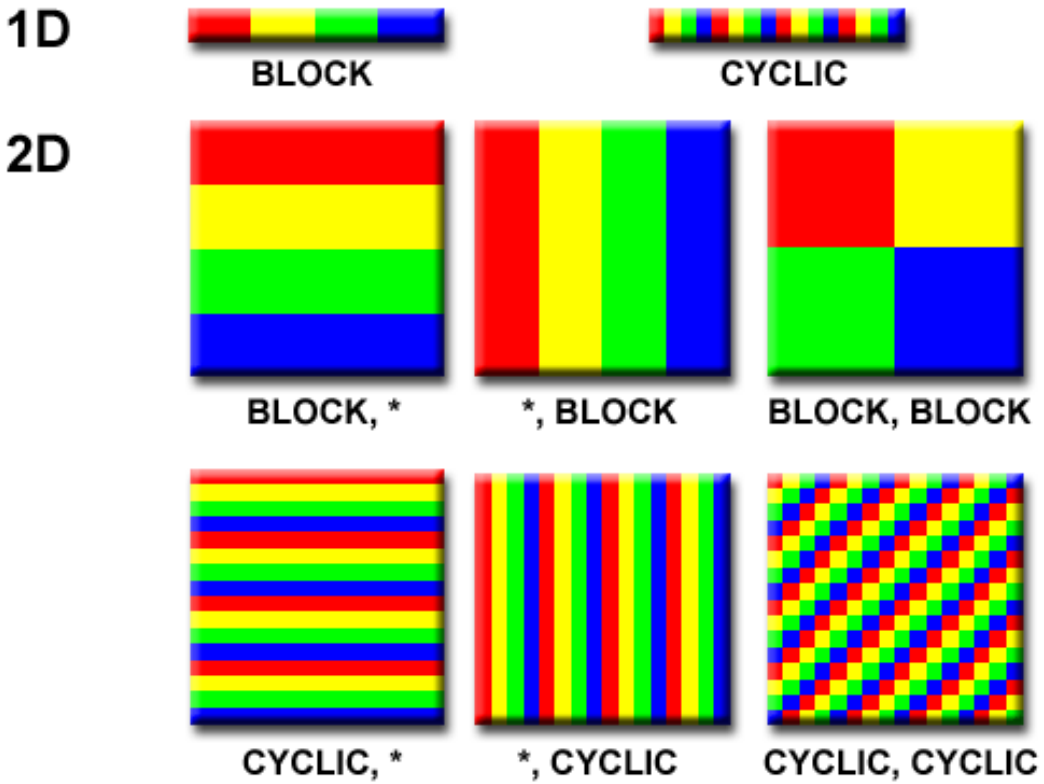
# Умножение на матрици

- Матриците имат регулярна структура
  - за паралелните изчисления с матрици и вектори се използва разделяне на данните (data decomposition)

- Обикновено паралелните алгоритми за умножение на матрица с вектор или матрица с матрица използват разделяне на входните, изходните или междинните данни

# Умножение на матрици

- Повечето алгоритми използват едномерно, двумерно блоково, циклично или блоково-циклично разделяне



# Умножение на матрици

- Сложността на последователния алгоритъм е $O(n^3)$
- Блоково разделяне на матрица
  - матрица $A$ с размери $n$ x $n$ се разглежда като съставена от масив с размери $q$ x $q$ от блокове $A_{i,j}$ ($0 \leq i, j < q$), такива че всеки блок е подматрица $(n/q)$ x $(n/q)$
- Умножението на матрици се свежда до $q^3$ матрични умножения на матрици $(n/q)$ x $(n/q)$

# Умножение на матрици

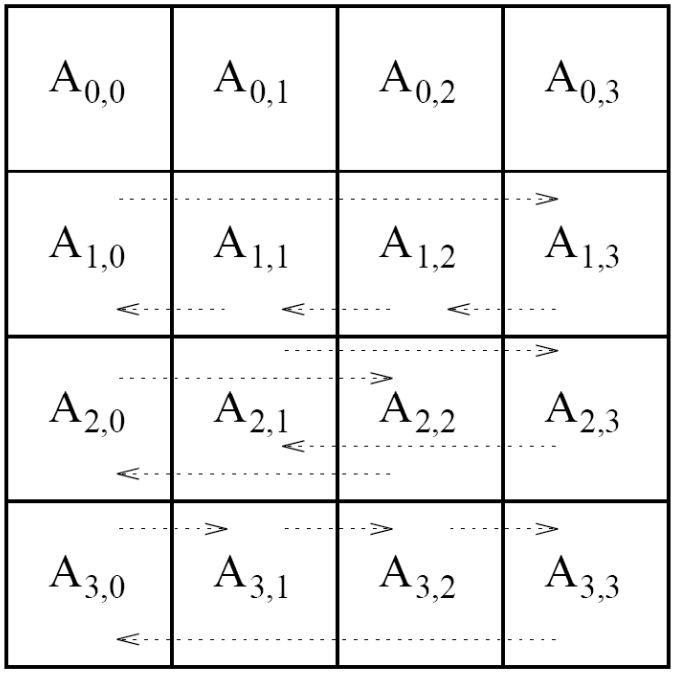
- Входните матрици $A$ и $B$ с размери $n$ x $n$ се разделят на $p$ блока $A_{i,j}$ и $B_{i,j}$ ($0 \leq i, j < \sqrt{p}$ ) с размер на всеки блок $(n/\sqrt{p}) \times (n/\sqrt{p})$
- Всеки процес първоначално съхранява $A_{i,j}$ и $B_{i,j}$
- Всеки процес извършва локално матрично умножение на подматриците за определяне на блок $C_{i,j}$ от резултантната матрица
  - за изчисляване на подматрицата $C_{i,j}$ са необходими всички подматрици $A_{i,k}$ и $B_{k,j}$ за $0 \leq k < \sqrt{p}$
  - използва с all-to-all broadcast на блоковете на матрицата $A$ по редове и на матрицата $B$ по колони

# Умножение на матрици

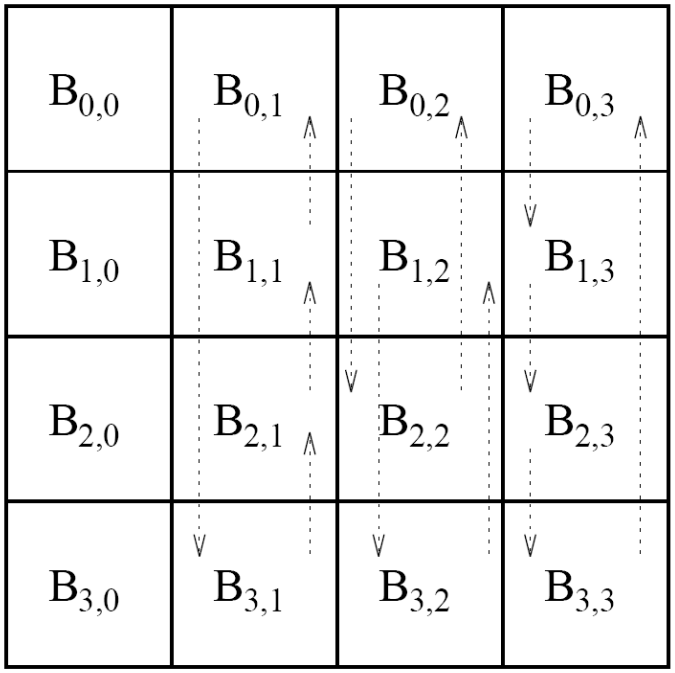
- Паралелният алгоритъм на Кенън (Cannon) за умножение на матрици използва декомпозиция на матриците на блокове
  - изискване на алгоритъма е броя процеси да бъде точен квадрат
    - при невъзможност да се спази това изискване могат да бъдат добавени фиктивни елементи в матриците
- Изчисленията се разпределят между процесорите така че всеки процес използва различен блок от матрицата A
- Блоковете систематично се ротират между процесите след всяко умножение на подматрици

# Умножение на матрици

- Начално разделяне на матриците A и B



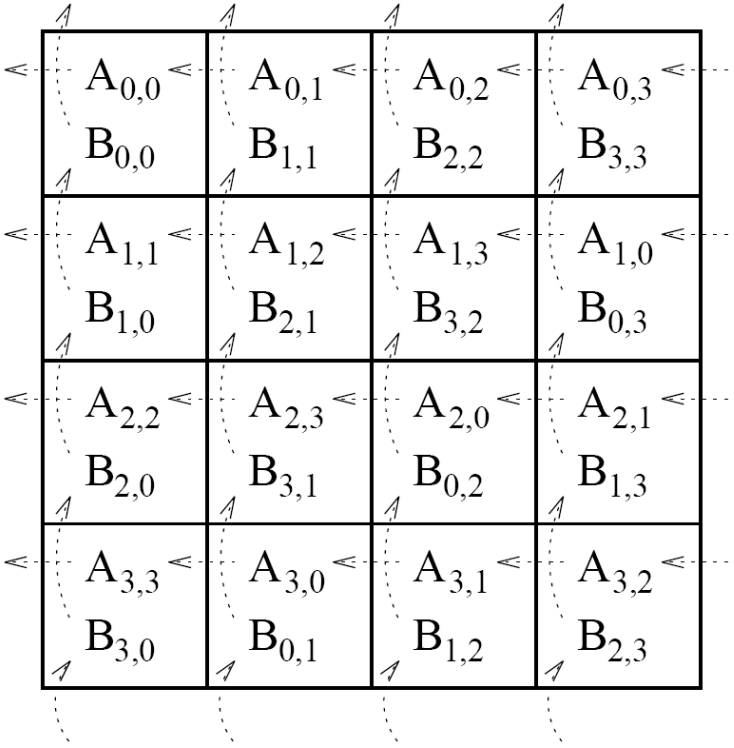

(a) Initial alignment of A



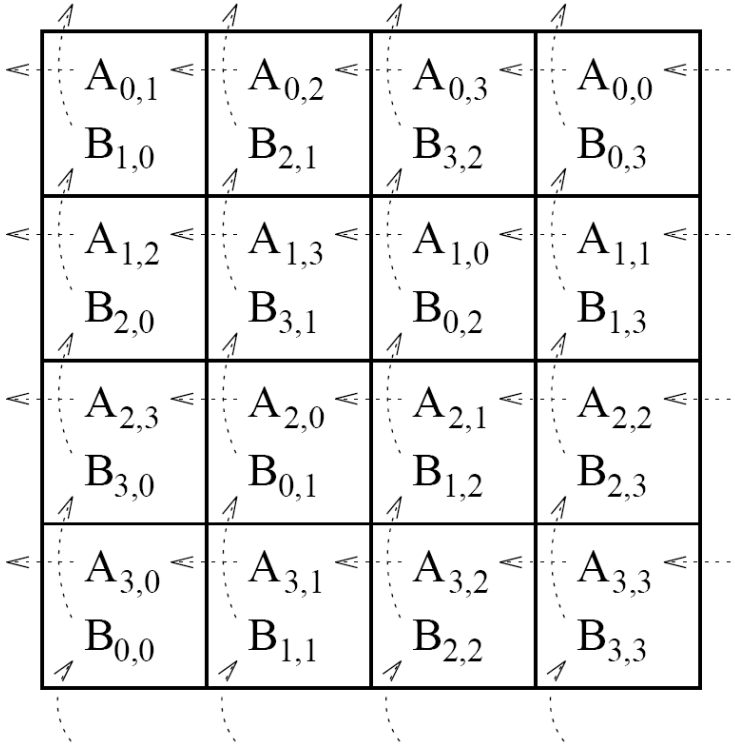

(b) Initial alignment of B

# Умножение на матрици

- Начално разделяне на матриците A и B и първо умножение на подматрици



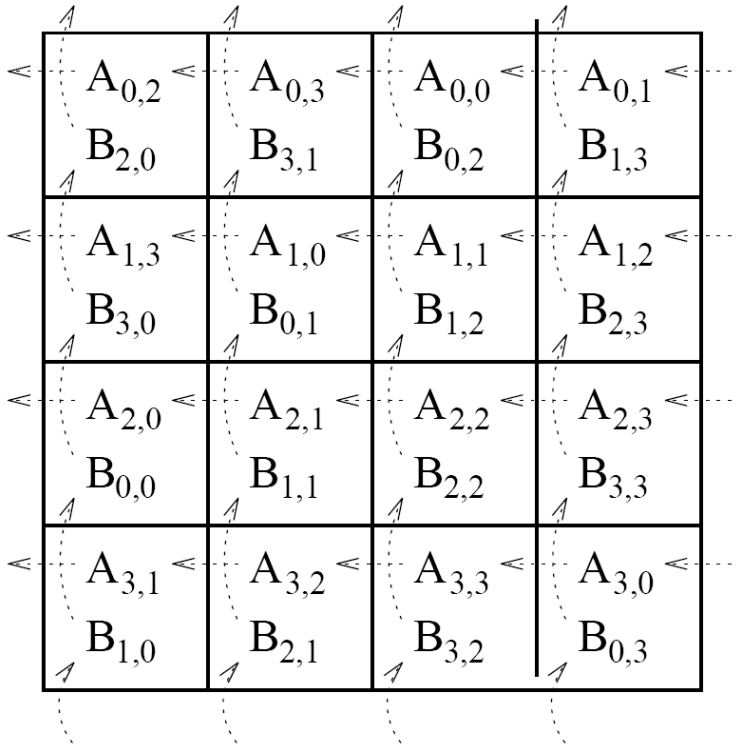

(c) A and B after initial alignment




(d) Submatrix locations after first shift

# Умножение на матрици

- Подматрици след второ и трето преместване

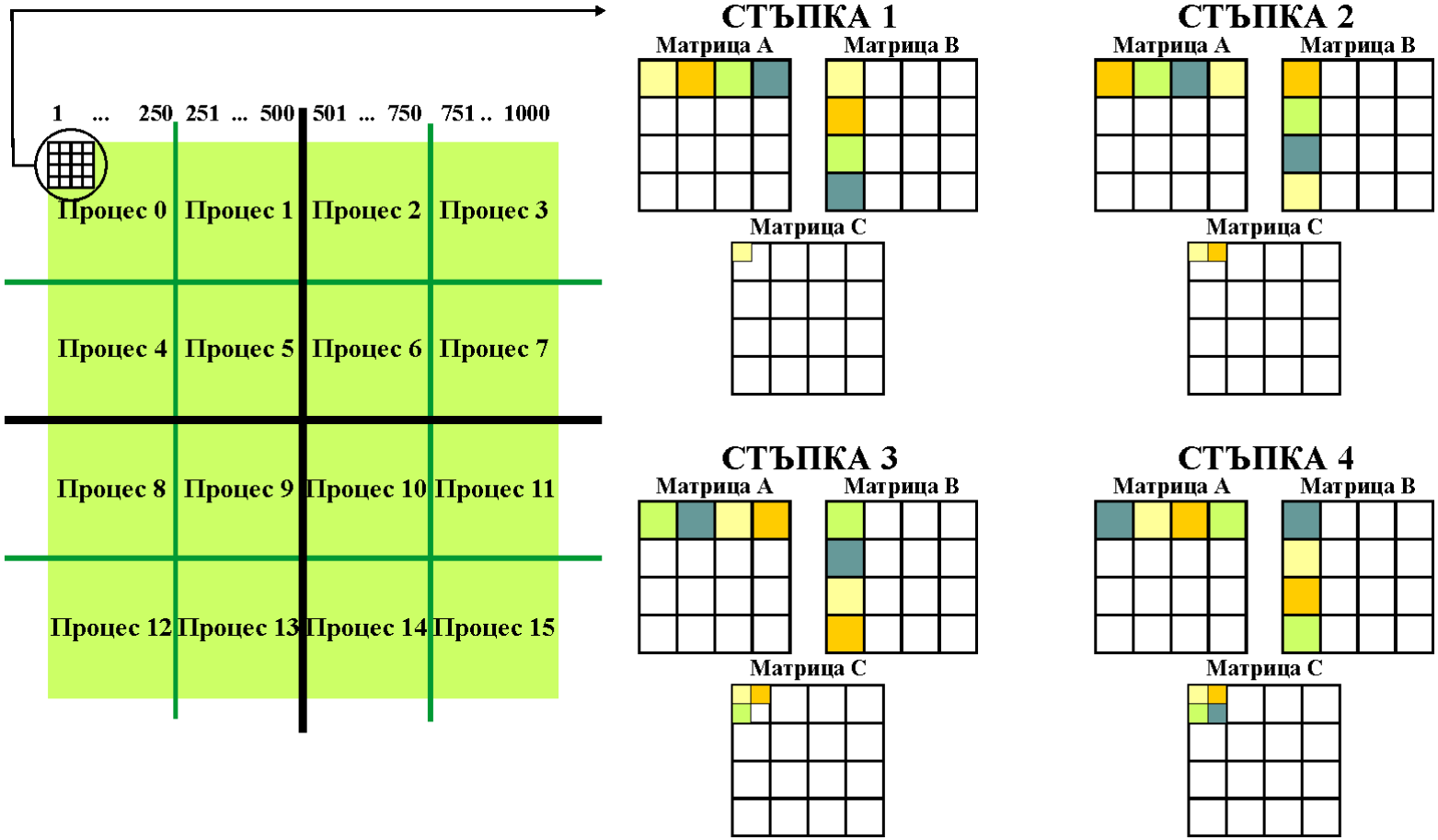



(e) Submatrix locations after second shift

| | | | |
|---|---|---|---|
| $A_{0,3}$ $B_{3,0}$ | $A_{0,0}$ $B_{0,1}$ | $A_{0,1}$ $B_{1,2}$ | $A_{0,2}$ $B_{2,3}$ |
| $A_{1,0}$ $B_{0,0}$ | $A_{1,1}$ $B_{1,1}$ | $A_{1,2}$ $B_{2,2}$ | $A_{1,3}$ $B_{3,3}$ |
| $A_{2,1}$ $B_{1,0}$ | $A_{2,2}$ $B_{2,1}$ | $A_{2,3}$ $B_{3,2}$ | $A_{2,0}$ $B_{0,3}$ |
| $A_{3,2}$ $B_{2,0}$ | $A_{3,3}$ $B_{3,1}$ | $A_{3,0}$ $B_{0,2}$ | $A_{3,1}$ $B_{1,3}$ |

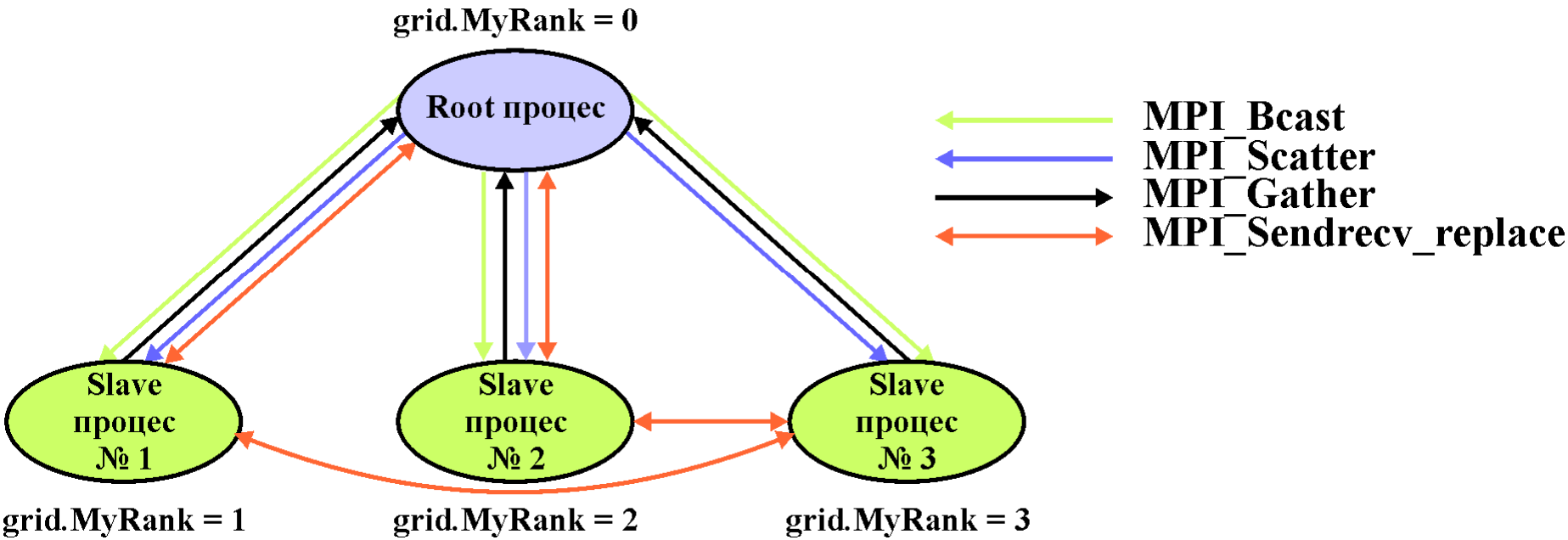
(f) Submatrix locations after third shift

# Умножение на матрици



# Умножение на матрици

- Използва се парадигмата синхронни итерации и мениджър-работници
- Процесите се организират в комуникатор с виртуална Декартова топология



# Умножение на матрици

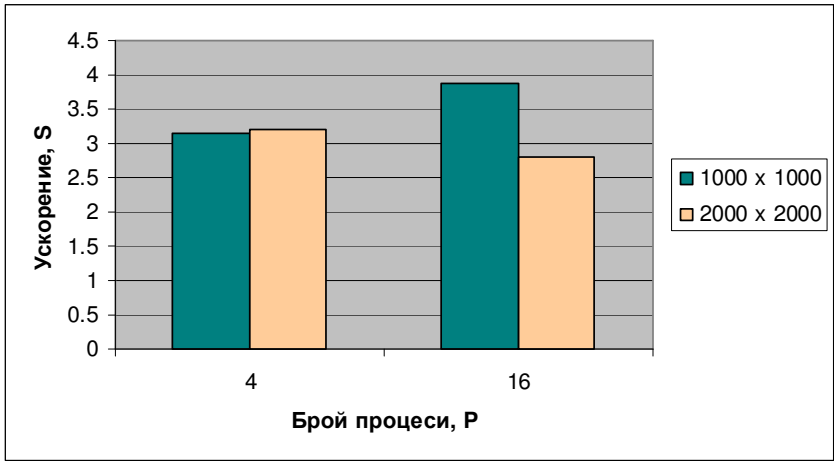
- Процесът-мениджър изпълнява следните дейности
  - Изпраща в режим на разпръскване размерите на умножаваните матрици до всички процеси-работници с *MPI_Bcast()*
  - Разпръсква подматриците на умножаваните матрици към всички процеси-работници с *MPI_Scatter()*, така че всеки процес-работник получава подматриците, върху които ще извършва изчисления
  - Получава от процесите-работници частичните резултати на изходната матрица с *MPI_Gather()*

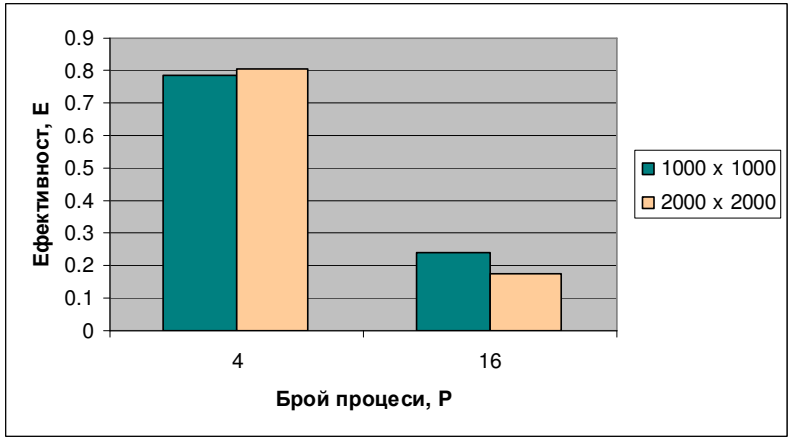
# Умножение на матрици

- Процесите-работници извършват следните дейности
  - Получават от процеса-мениджър размерите на умножаваните матрици с *MPI_Bcast()*
  - Получават от процеса-мениджър подматриците, върху които ще работят с *MPI_Scatter()*
  - Извършват изчисленията за умножаване на подматриците
  - Изпращат частичните резултати на процеса-мениджър с *MPI_Gather()*
  - Изпращат на съседните процеси подматриците, които вече са обработили
    - едната матрица се изпраща на левия съседен процес в реда
    - другата на горния съседен процес в колоната съгласно виртуалната Декартова топология на процесите в комуникатора
    - използва се функцията *MPI_ Sendrecv_replace()*
  - Получават от съседните процеси нови подматрици за умножение на следващата стъпка с *MPI_ Sendrecv_replace()*

# Умножение на матрици

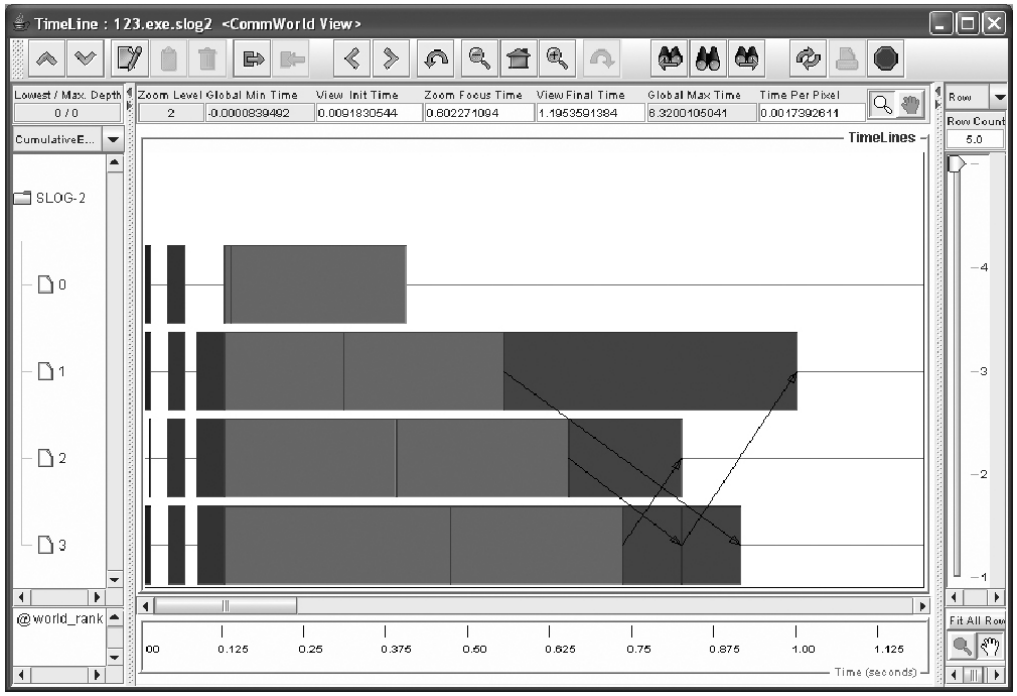
- Производителност
  - добро ускорение и утилизация на процесорите
  - не се мащабира линейно поради интензивни комуникации на всяка итерация



# Умножение на матрици

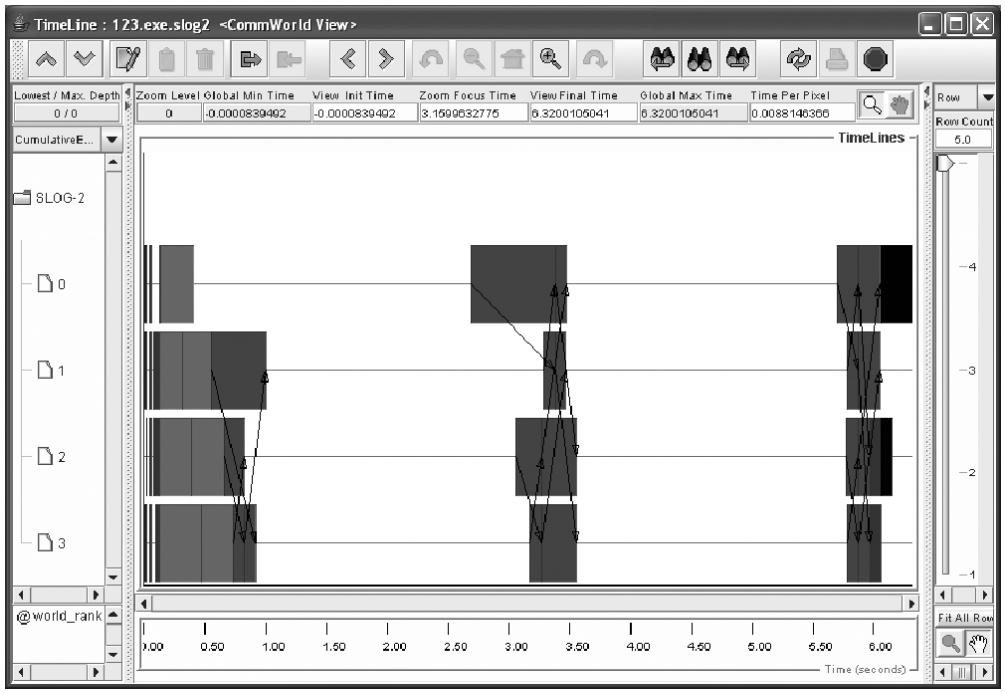
- Профил на паралелизмите и диаграма на Гант




Първоначално разпределяне на матриците между подчинените процеси и първа итерация на алгоритъма

# Умножение на матрици

- Профил на паралелизмите и диаграма на Гант



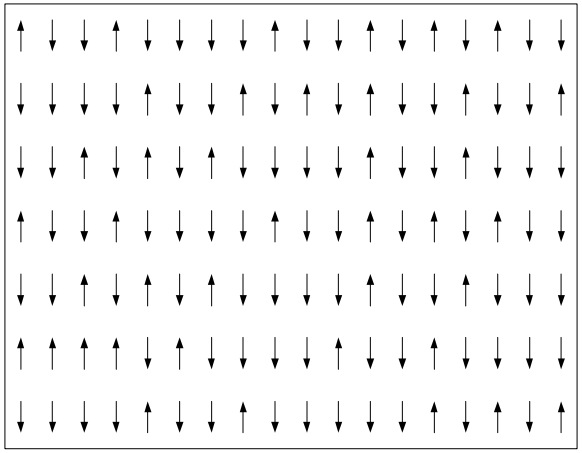

Размяна на подматрици след итерация на локално умножение

# Изинг моделиране на прост магнит

- *Задача*
  - да се симулира енергийното състояние на прост магнит с използване на двумерен Изинг модел
- *Целева паралелна компютърна платформа*
  - клъстер от многоядрени възли
- *Програмен модел*
  - Хибриден (обмен на съобщения + многонишкова обработка)
- *Имплементация*
  - MPI + OpenMP

# Изинг моделиране на прост магнит

- Изинг модел
  - описва състоянието на двумерна решетка, съставена от клетки (магнитни спинове)
  - всяка клетка може да бъде в едно от две възможни състояния
  - енергията на системата се определя от сумата на елементарните взаимодействия между всеки спин и неговите съседи в решетката
  - взаимодействието между зарядите води до промяна на стойностите на някои от тях
  - енергията на системата е функция на състоянията на спиновете



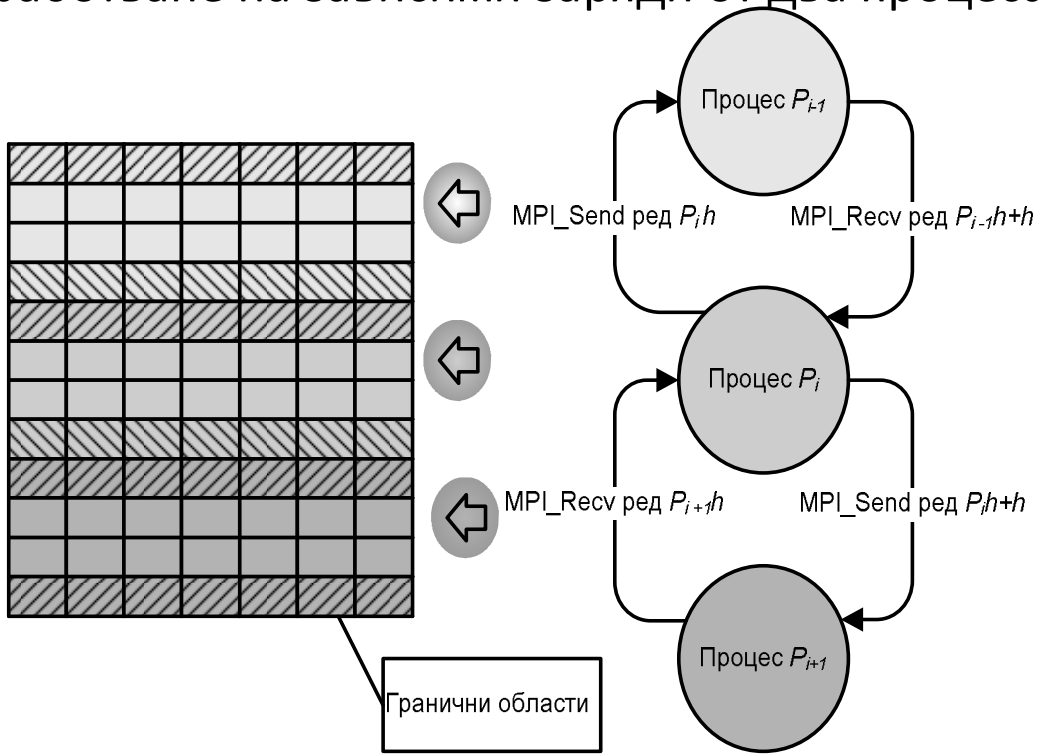
- Алгоритъмът Метрополис
  - генериране на последователност от случайни проби при дадена вероятностна функция на разпределение
  - следващото състояние се генерира с определена вероятност на база на текущото състояние

# Изинг моделиране на прост магнит

- Алгоритъм Метрополис за симулиране на прост магнит с Изинг модела
  - регулярност и локалност на изчисленията
- Регулярността
  - лесно паралелизиране с използване на декомпозиция по данни и добро балансиране на товара
- Локалност
  - локалност на комуникациите и лесна синхронизация на работата на процесите
- Декомпозиция на данните
  - лентова декомпозиция за разпределяне на решетката между процесите
  - при решетка с размери $N \times N$ всеки процесор обработва част от решетката с размери $h \times N$, където $h = N/p$, а $p$ е броят на процесите, участващи в паралелната обработка

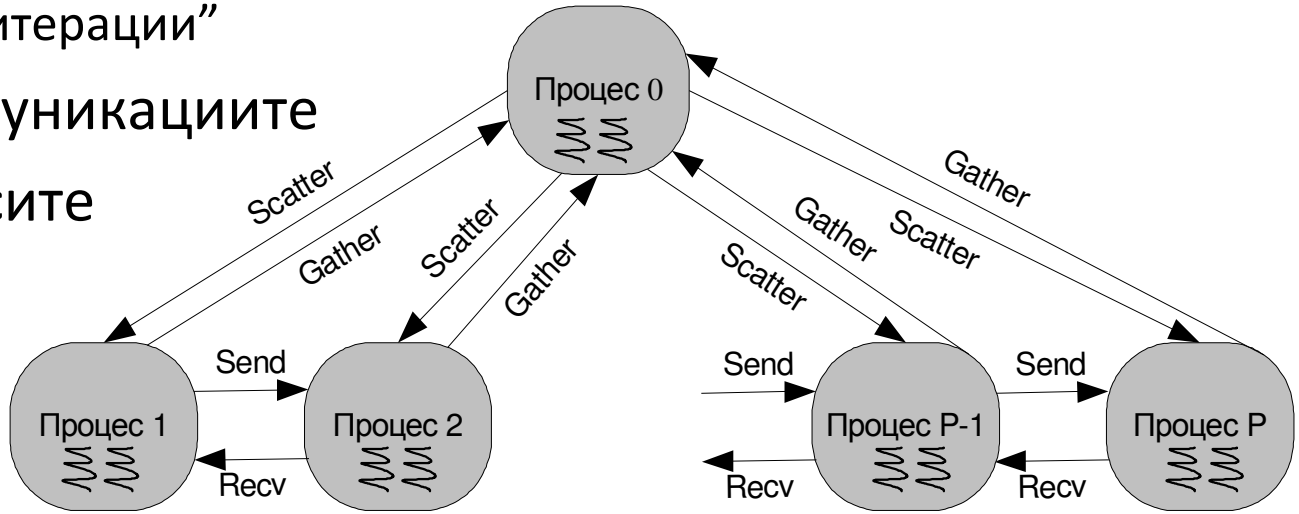
# Изинг моделиране на прост магнит

- Процесите комуникират помежду си за обменяне на необходимите им данни от граничните области
  - комуникацията се налага за предотвратяване на едновременното обработване на зависими заряди от два процеса



# Изинг моделиране на прост магнит

- Хибриден програмен модел
  - множество процеси, комуникиращи чрез обмен на съобщения
  - всеки процес използва многонишкова обработка
- Алгоритмична парадигма
  - „мениджър-работници”
  - “синхронни итерации”
- Модел на комуникациите между процесите



# Изинг моделиране на прост магнит

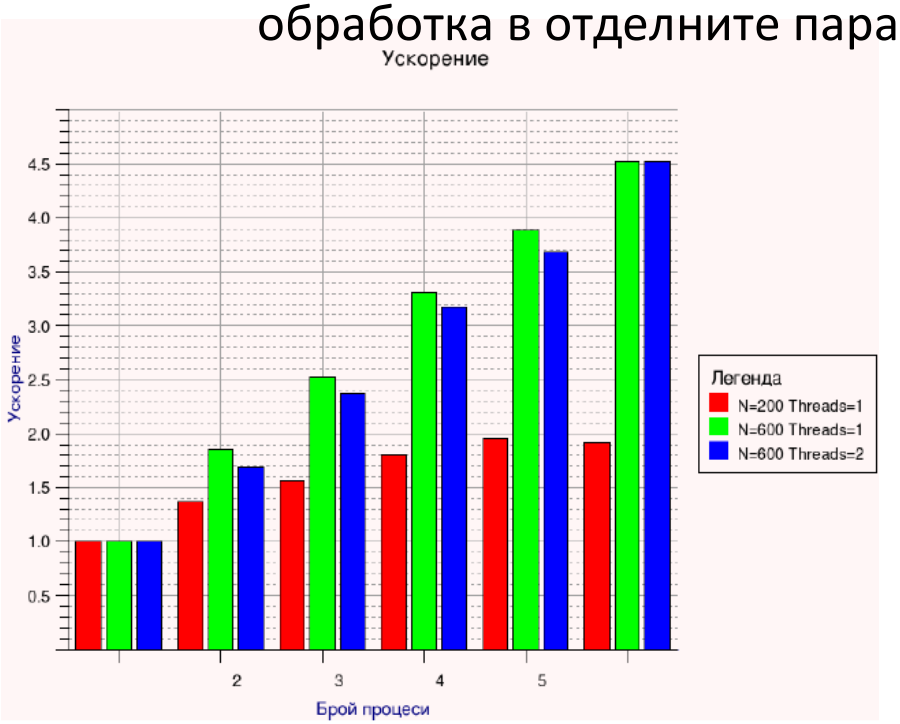
- Процесът-мениджър изпълнява следните дейности
  - Генерира случайно начално състояние на решетка със зададени размери
  - Разделя решетката на ленти съобразно броя процеси и разпределя данните към процесите-работници с функцията *MPI_Scatter()*
  - Обработва своята част от решетката изпълнявайки съответните операции като процес-работник
  - Събира от процесите-работници получените данни и ги обединява за получаване на текущото състояние на решетката – използва се функцията *MPI_Gather()*

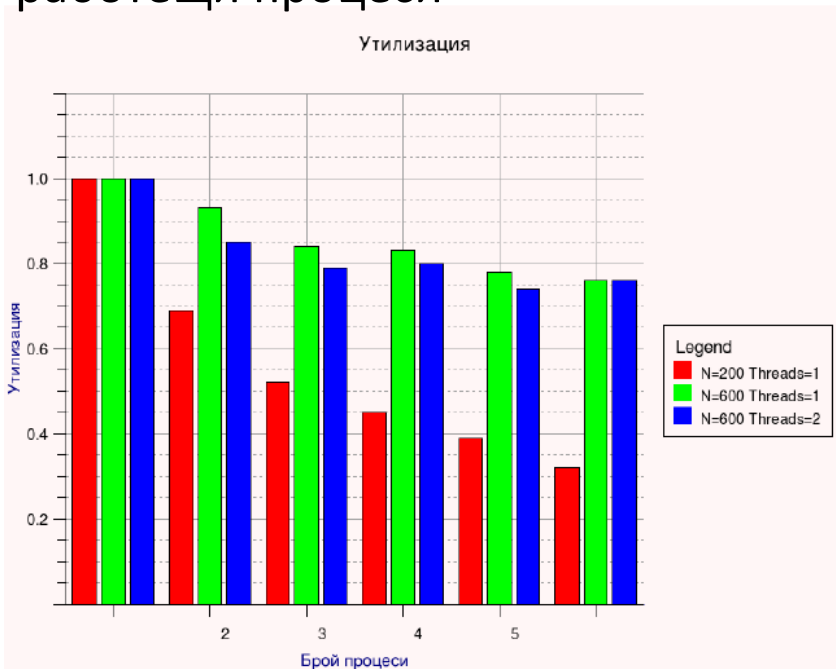
# Изинг моделиране на прост магнит

- Процесът-работник с ранг $P_i$ извършва следните операции
  - Получава от процеса-мениджър данните за своята част от решетката чрез обръщение към функцията за колективна комуникация *MPI_Scatter()*
  - Обработва данните от неговата лента $P_{i.h}$ до $P_{i.h+h}$ като променя състоянията на клетките съгласно двумерния изинг модел и алгоритъм Метрополис
  - Изпраща ред $P_{i.h}$ на процес с ранг $P_{i-1}$ с функцията *MPI_Send()*
  - Получава ред $P_{i+1.h}$ от процес с ранг $P_{i+1}$ с функцията *MPI_Recv()*
  - Изпраща ред $P_{i.h+h}$ на процес с ранг $P_{i+1}$ с функцията *MPI_Send()*
  - Получава ред $P_{i.h}$ от процес с ранг $P_{i-1}$ с функцията *MPI_Recv()*
  - Изпраща на процеса-мениджър обработените данни за неговата част от решетката чрез обръщение към функцията за колективна комуникация *MPI_Gather()*

# Изинг моделиране на прост магнит

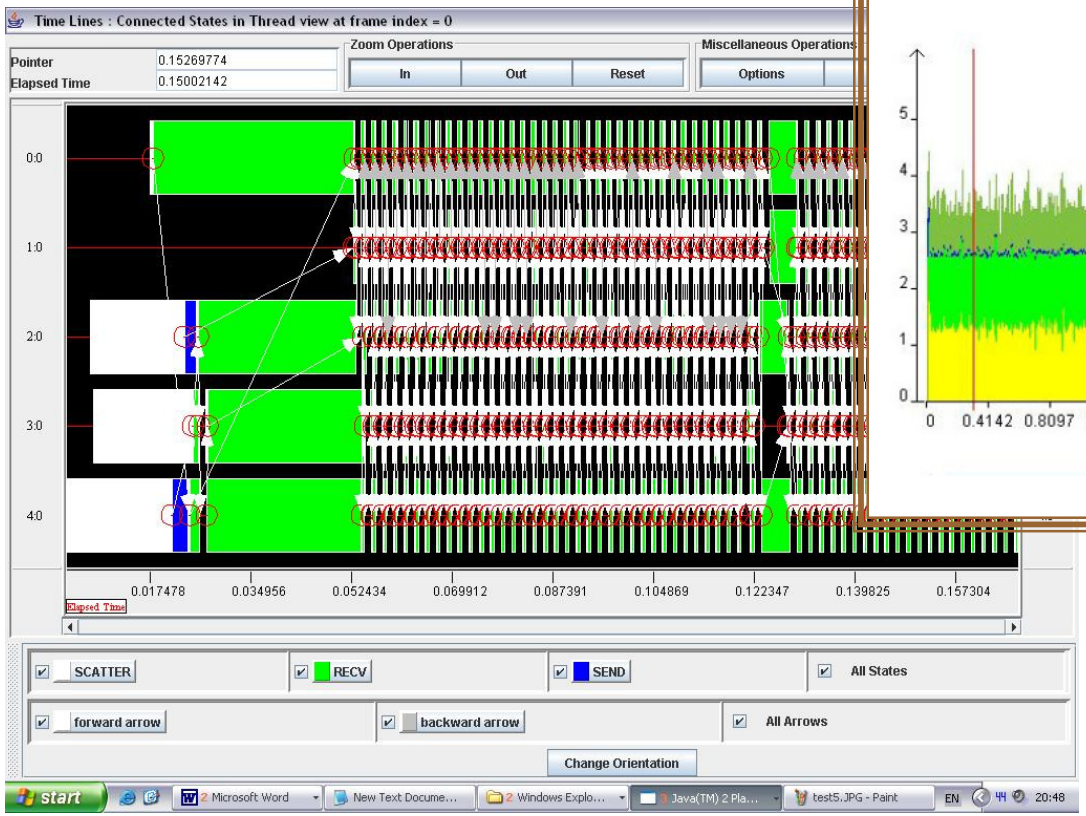
- Производителност
  - почти линейно ускорение
  - висока, почти постоянна утилизация на процесорите
  - допълнително увеличаване на ускорението при многонишкова обработка в отделните паралелно работещи процеси

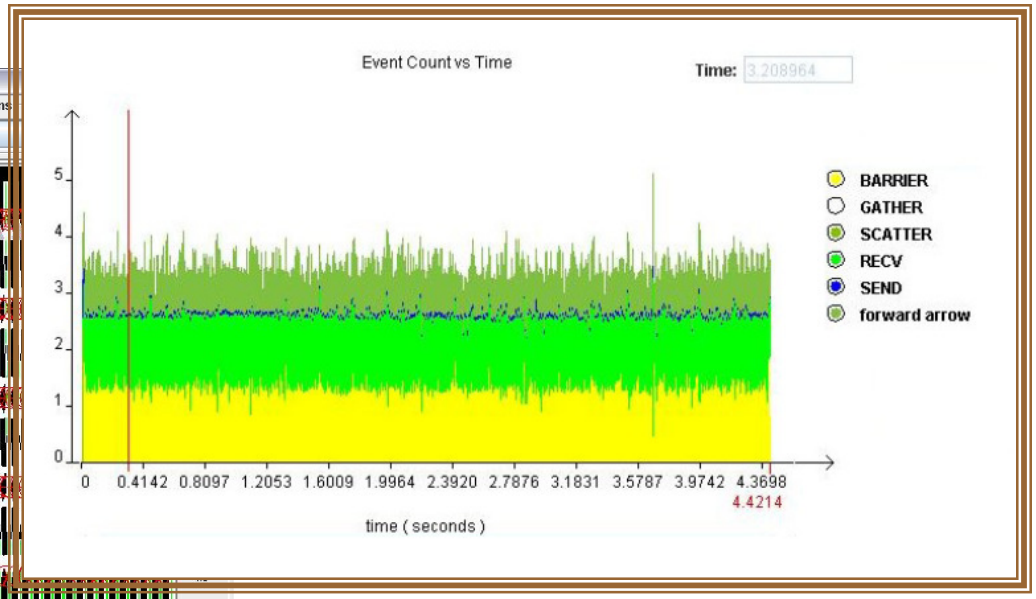


# Изинг моделиране на прост магнит

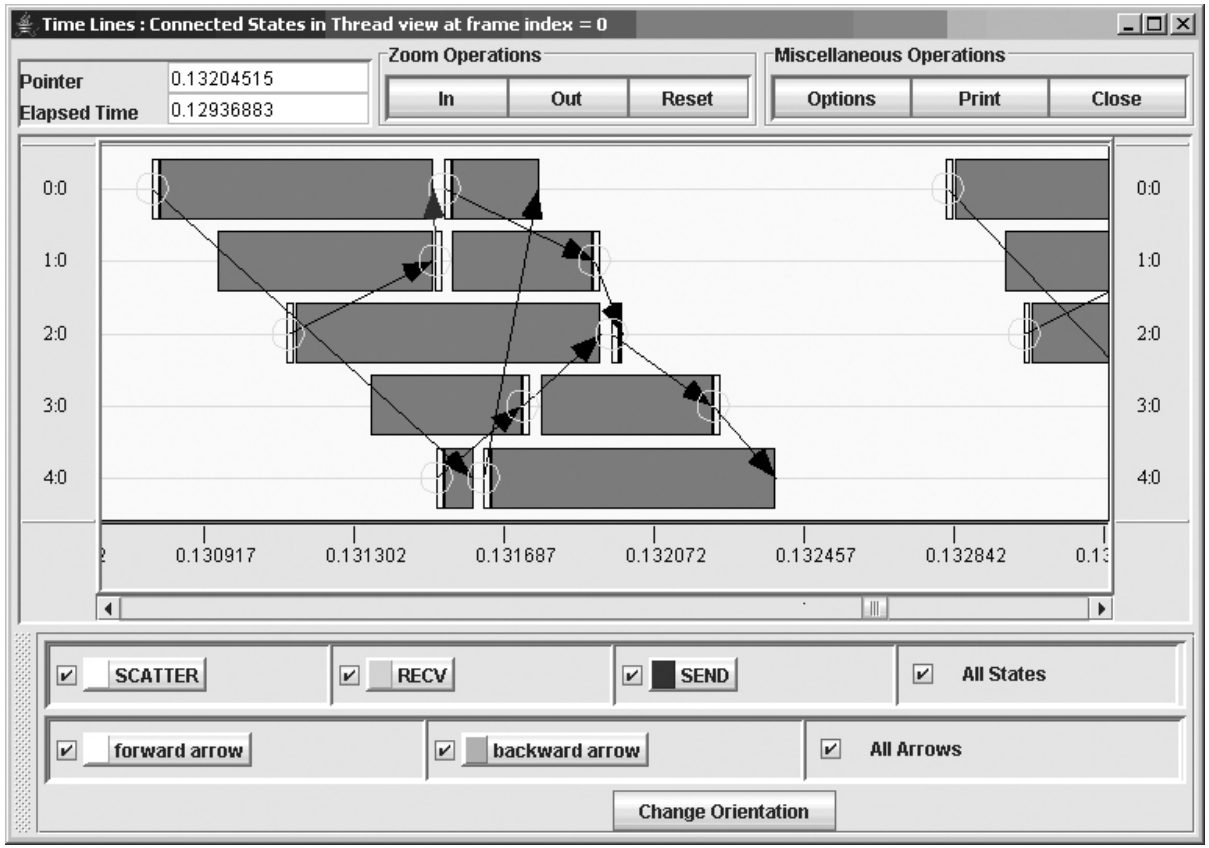
- Профил на паралелизмите и диаграма на Гант
  - 5 процеса, размер на решетката 200×200, 15 000 итерации,



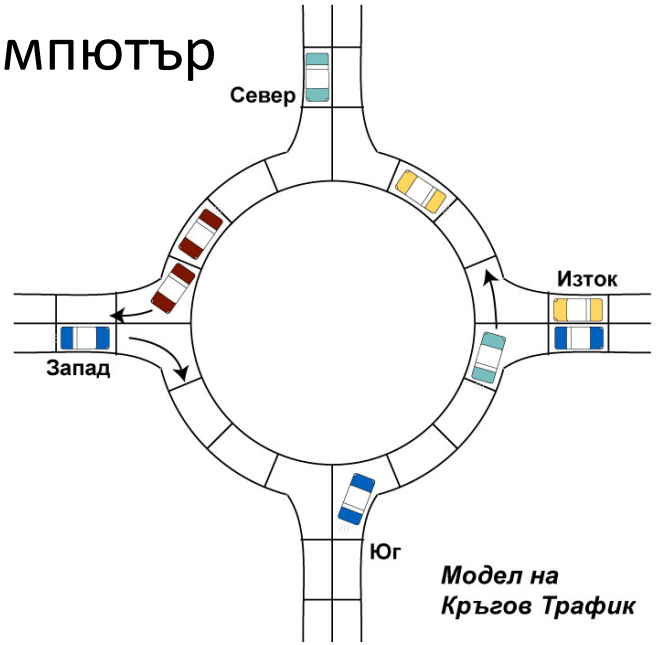
# Изинг моделиране на прост магнит

- Комуникация между процесите за обмен на граничните стойности



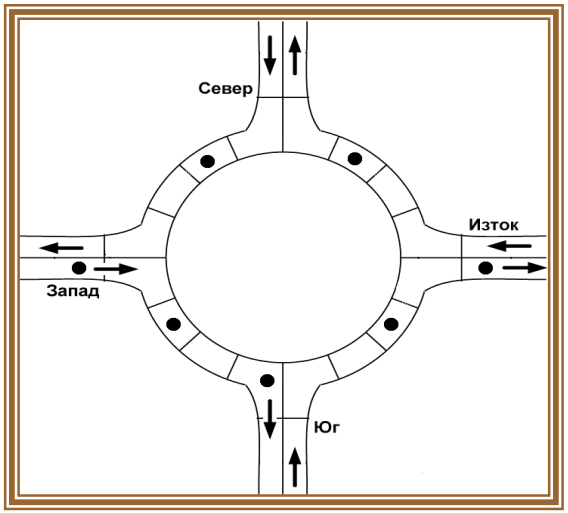
# Симулация на кръгов трафик

- *Задача*
  - да се симулира движение на автомобили в кръгов трафик
- *Целева паралелна компютърна платформа*
  - многопроцесорен/многоядрен компютър
- *Програмен модел*
  - Многонишков
- *Имплементация*
  - OpenMP



# Симулация на кръгов трафик

- Симулационен модел на кръгово движение
  - симулира придвижването на моторни превозни средства в рамките на кръгов трафик
  - предварително зададени вероятност за пристигане на превозни средства на всяка от входните точки
  - вероятности за напускане на превозни средства през един от изходите на кръга
  - включване на автомобилите в трафика през една от четири възможни посоки
  - всяко превозно средство се движи около кръга в посока обратна на часовниковата стрелка
  - превозните средства в кръга имат предимство пред тези, които пристигат на някой от входовете



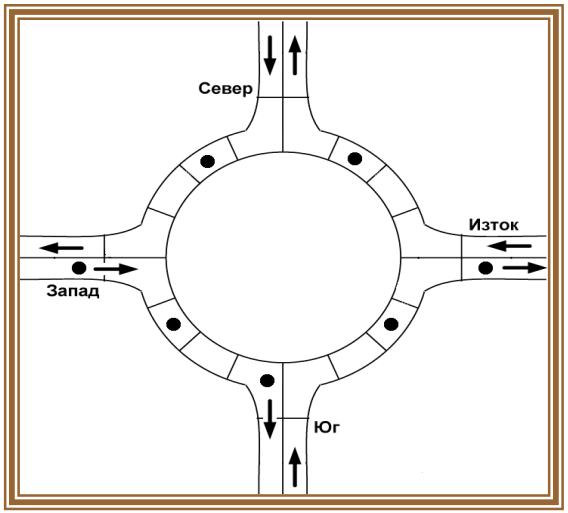

| | С | З | Ю | И | |
|---|---|---|---|---|---|
| 0 | 0 | 4 | 8 | 12 | offset |
| 1 | 1 | 0 | 0 | 1 | arrival |
| 26 | 26 | 23 | 22 | 38 | arrivalCount |
| 19 | 19 | 13 | 11 | 26 | waitCount |
| 1 | 1 | 2 | 0 | 3 | Queue |
| 37 | 37 | 49 | 20 | 41 | QueueAccum |

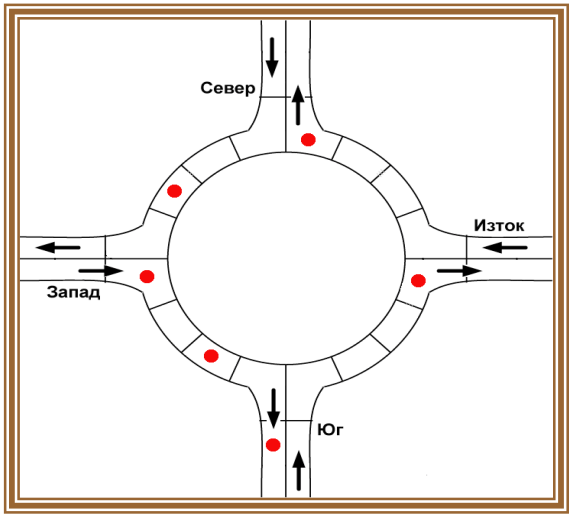
| 71 |
|---|
| итерация |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | -1 | 4 | -1 | 8 | 8 | 12 | 0 | -1 | -1 | 12 | 12 | 8 | 0 | 0 | 0 |

# Симулация на кръгов трафик

- Кръговото движение се представя с клетъчен автомат чрез разделяне на кръга на краен брой клетки (секции)
  - в даден момент от времето всяка секция може да бъде свободна или заета от не повече от едно превозно средство
  - на всяка стъпка от симулацията превозните средства се придвижват с една клетка по посока обратна на часовниковата стрелка или напускат кръга през един от четирите изхода
- Цел на симулацията
  - да се намерят оценки на вероятностите автомобилите да чакат при пристигане на всеки от входовете на кръговото движение за присъединяване към трафика
  - средна дължина на опашките от автомобили, чакащи за присъединяване към трафика на всеки един от входовете



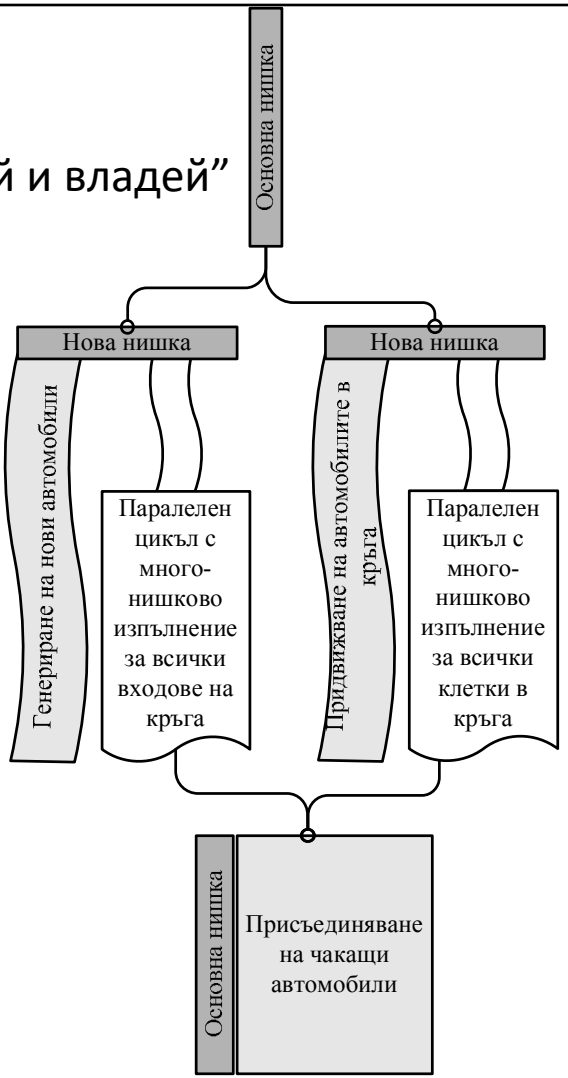



# Симулация на кръгов трафик

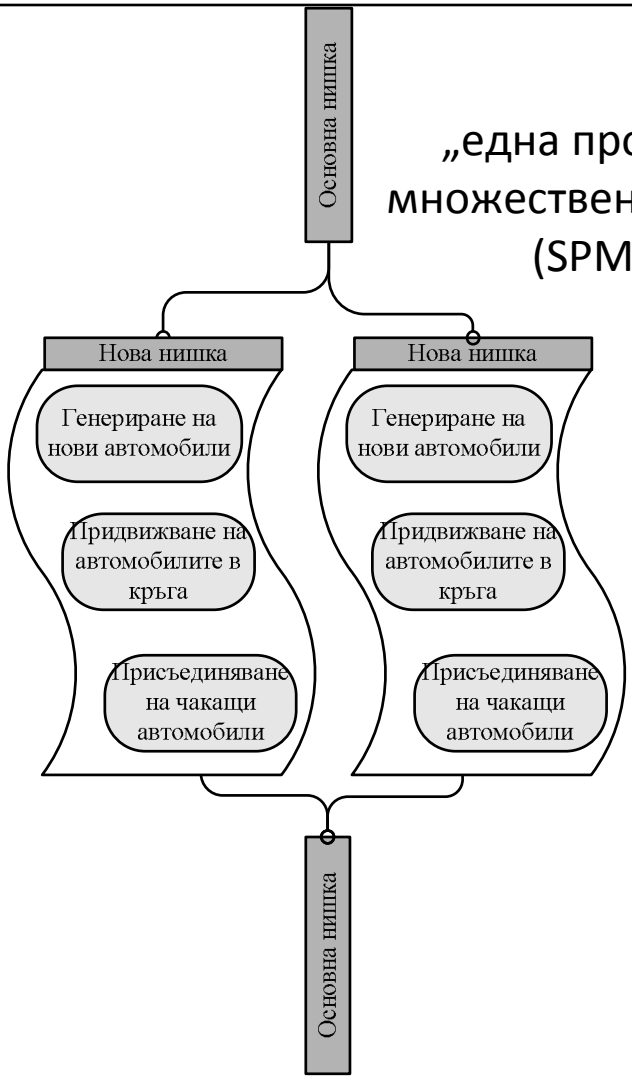
- Две алгоритмични парадигми
  - „разделяй и владей”
    - функционална декомпозиция
    - паралелни региони - едновременно изпълнение на няколко функции от различни процесорни нишки
    - допълнителен паралелизъм - многонишково изпълнение на паралелни цикли във всяка отделна функция
  - „една програма множествени данни” (SPMD)
    - декомпозиция на данните
    - масивите за съхраняване на необходимите за симулацията данни се разделят и се обработват едновременно от няколко паралелни нишки
    - разделянето е на равни части в зависимост от броя процесори в многопроцесорната система

# Симулация на кръгов трафик

„разделяй и владей”

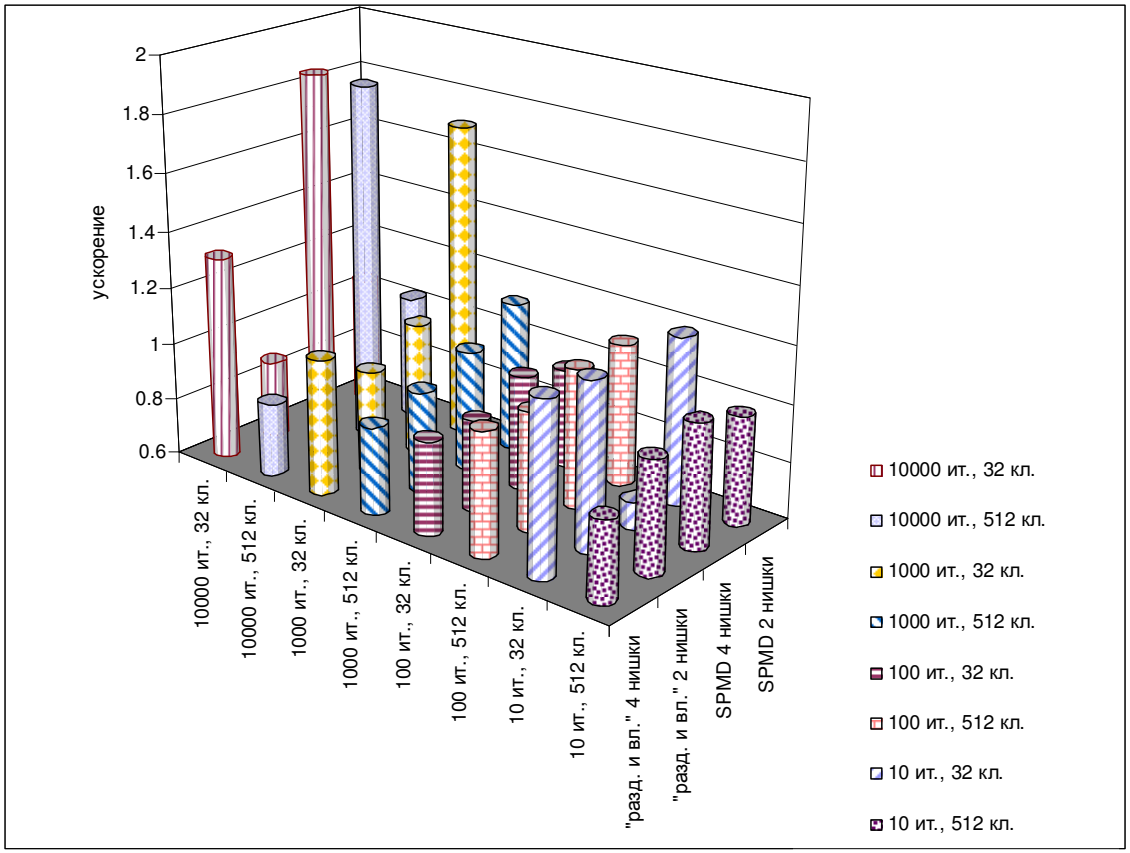



„една програма множествени данни” (SPMD)



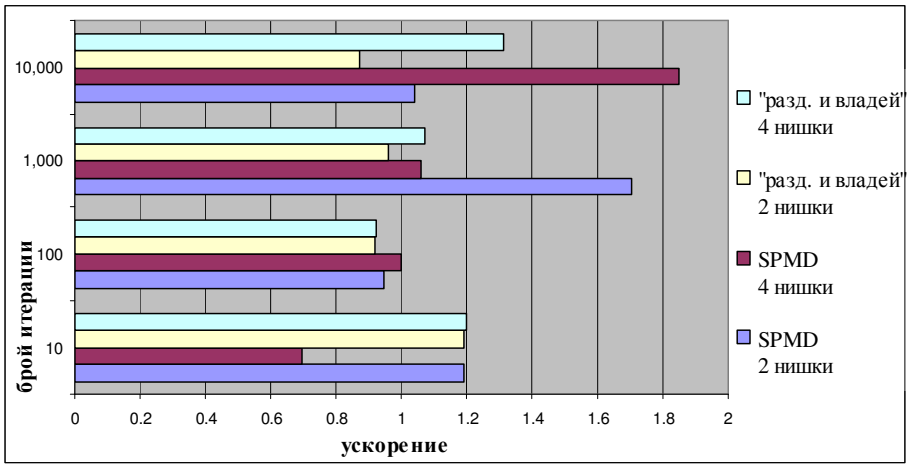
# Симулация на кръгов трафик

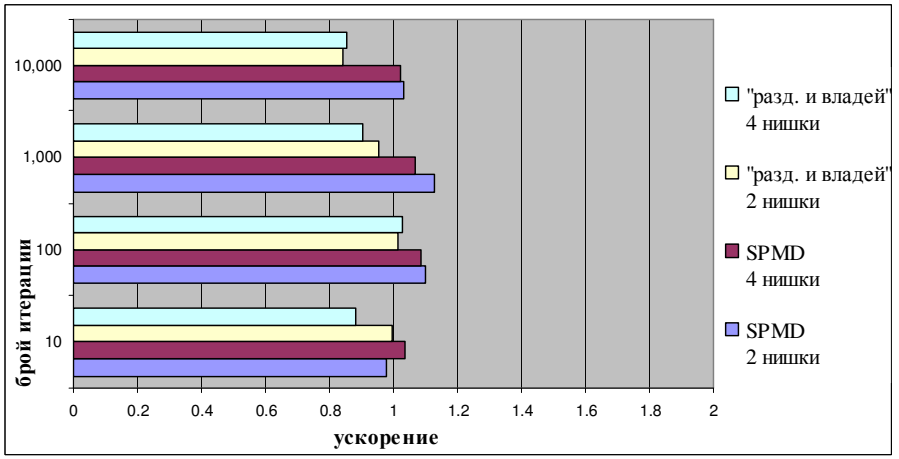
- Производителност
  - ускорение



# Симулация на кръгов трафик

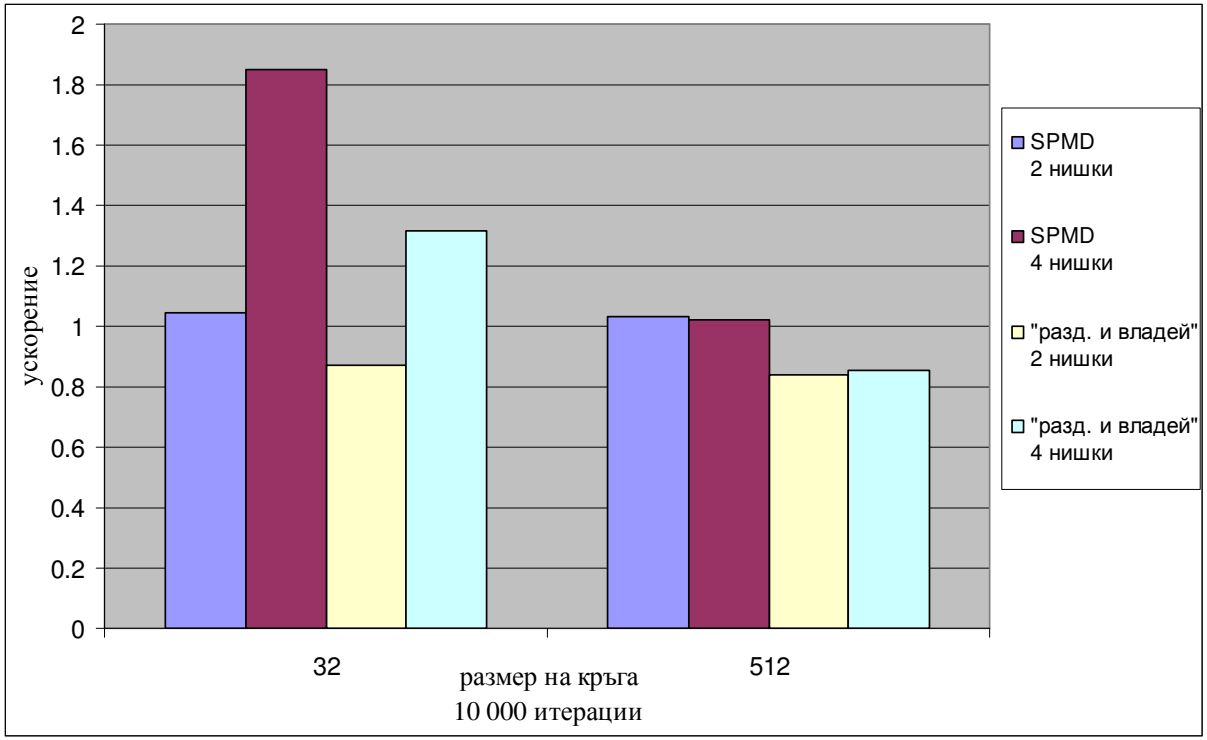
- Производителност
  - ускорение – мащабиране на данните



# Симулация на кръгов трафик

- Производителност
  - ускорение



# Симулация на кръгов трафик

- Производителност
  - утилизация