

پردازش موازی با

حمید نادران طحان

عضو هیئت علمی دانشگاه صنعتی امیرکبیر

دانشکده مهندسی مکانیک

۲۹ مهر ماه ۱۳۹۴

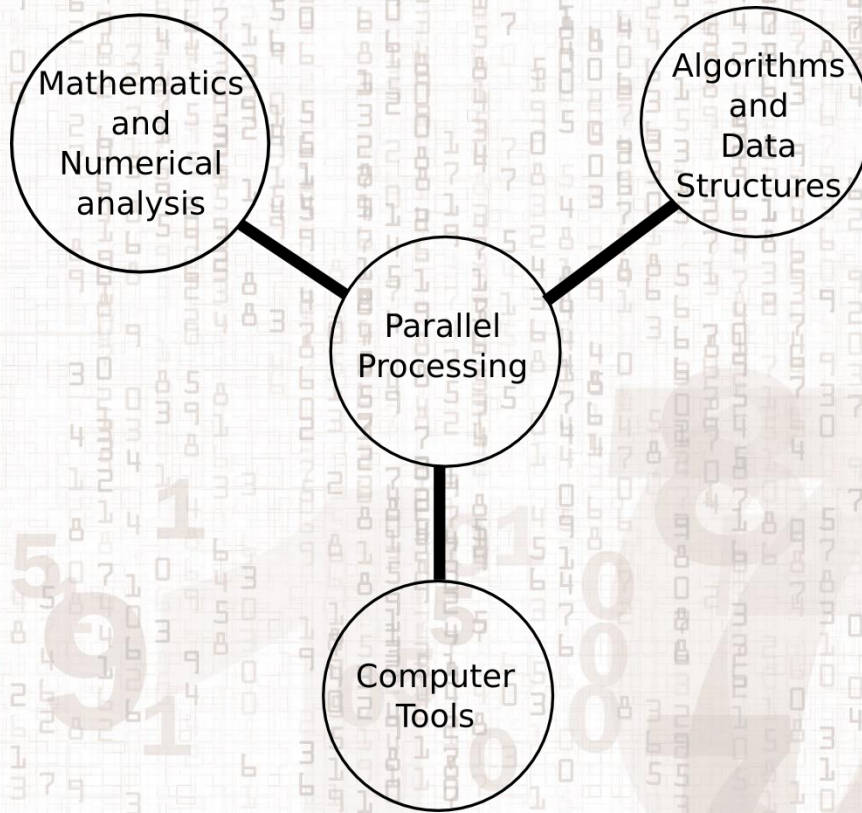
عناوین

- مقدمه
- تاریخچه
- تغییرات روندها
- وضعیت فعلی
- اصول
 - ملاحظات سخت افزاری
 - ملاحظات نرم افزاری
 - افزایش سرعت و مقیاس پذیری
- ابزارها
 - OpenMP
 - MPI
- نکات تکمیلی
- جمع بندی
- معرفی کارگاه آموزشی

ف:

پردازش موازی چیست؟

توانایی بهره گیری از چندین واحد پردازشی
به صورت همزمان برای انجام محاسبات



تاریخچه

۱۹۵۰

۱۹۶۰

۱۹۷۰

۱۹۸۰

۱۹۹۰

۲۰۰۰

بحث های اولیه

- ۱۹۵۵ - طراحی IBM 704 توسط امدال

- ۱۹۵۸ - مطرح شدن بحث استفاده همزمان از چند کامپیوتر توسط کاک و اسلاتنیک



تاریخچه

۱۹۵۰ ۱۹۶۰ ۱۹۷۰ ۱۹۸۰ ۱۹۹۰ ۲۰۰۰

معرفی سیستم های چند پردازنده ای



• ۱۹۶۲ - D825 با چهار پردازنده توسط بروز

• ۱۹۶۴ - CDC 6600 با ده پردازنده

• ۱۹۶۷ - مطرح شدن محدودیت افزایش سرعت
توسط امدال

تاریخچه

۱۹۵۰ ۱۹۶۰ ۱۹۷۰ ۱۹۸۰ ۱۹۹۰ ۲۰۰۰



گسترش سریع



• ۱۹۷۱ - معرفی 4004 توسط اینتل

• ۱۹۷۲ - تاسیس شرکت کری توسط سیمور کری

• ۱۹۷۴ - معرفی CDC STAR-100

اولین پردازنده برداری

• ۱۹۷۸ - بحث در مورد زبان برنامه نویسی موازی

تاریخچه

۱۹۵۰ ۱۹۶۰ ۱۹۷۰ ۱۹۸۰ ۱۹۹۰ ۲۰۰۰

دوران طلایی سوپر کامپیوترها

- ۱۹۸۰ - توسعه الگوریتم های انتقال پیام
- ۱۹۸۴ - استفاده از سیستم عامل موازی در CRAY - XMP
- ۱۹۸۷ - اختصاص جایزه گوردون-بل برای ۶۰۰ برابر افزایش سرعت
- ۱۹۸۸ - رشد سیستم های تک پردازنده ای



تاریخچه



تاریخچه

۱۹۵۰ ۱۹۶۰ ۱۹۷۰ ۱۹۸۰ ۱۹۹۰ ۲۰۰۰



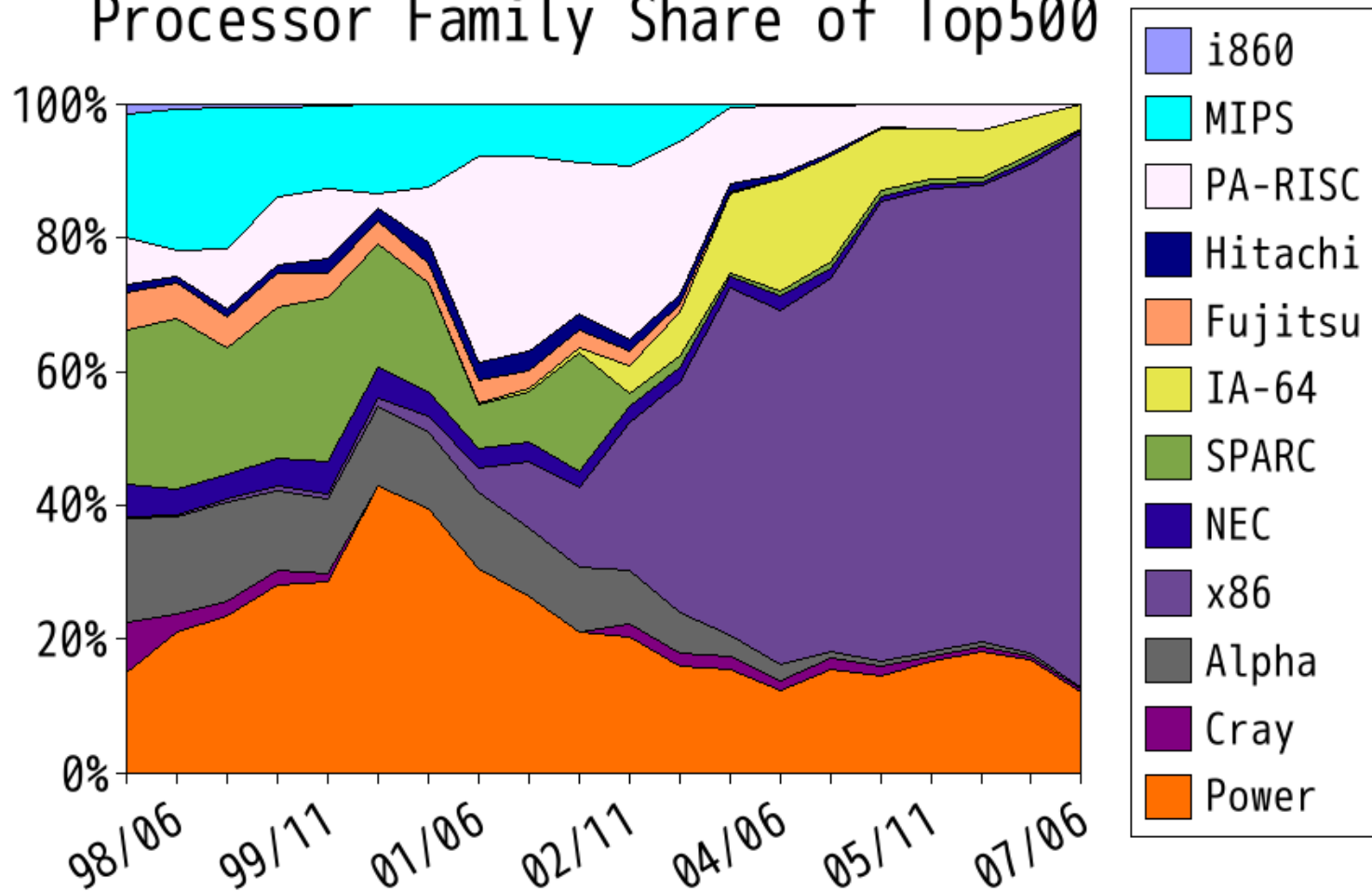
۱
معکوس شدن روند



- ۱۹۹۰ - پردازش برداری در تک پردازنده ها
- ۱۹۹۴ - طراحی اولین کلاستر بیولف توسط بکر و استرلینگ
- ۱۹۹۵ - تعطیل شدن بسیاری از شرکت های سازنده سوپر کامپیوتر

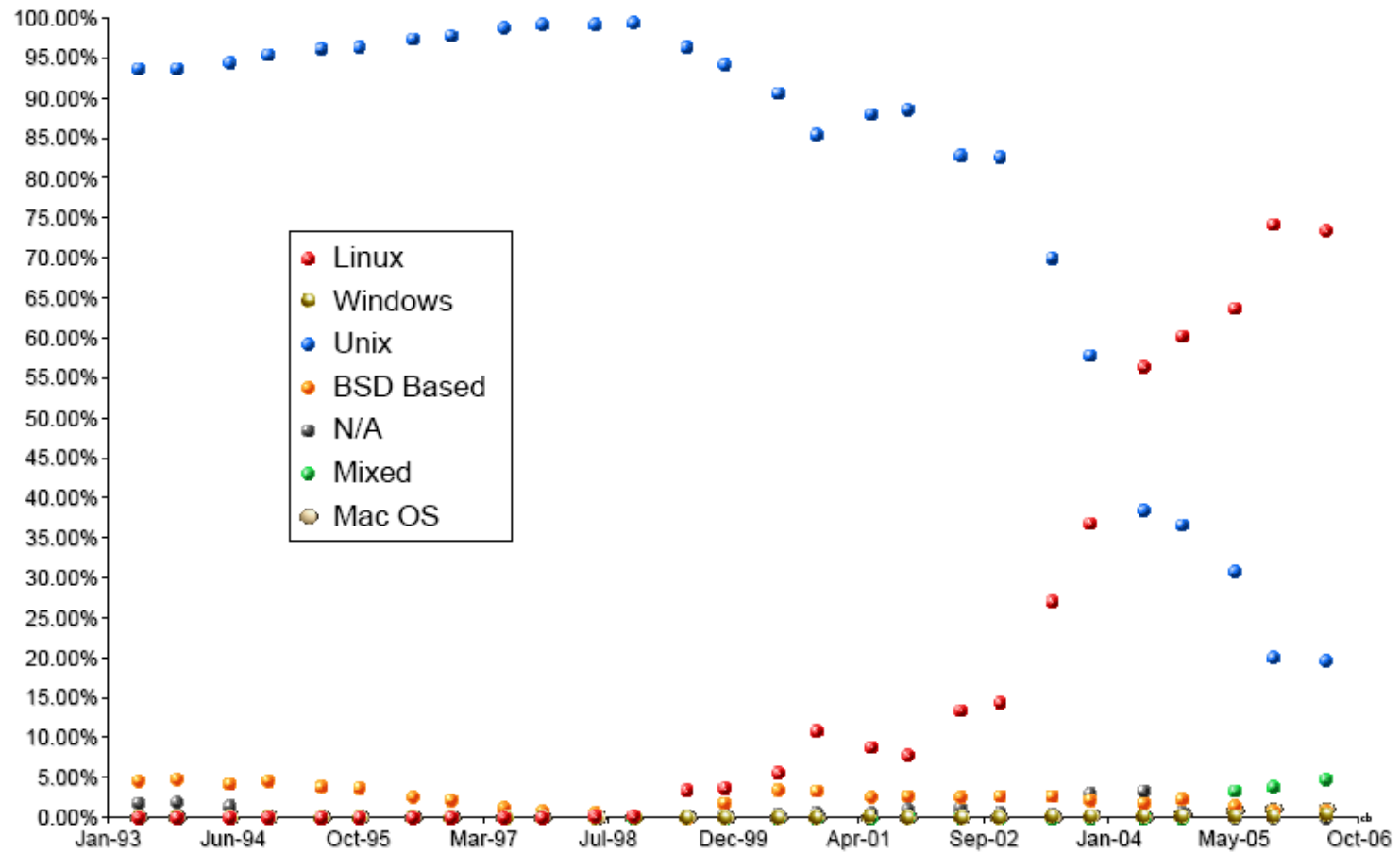
تغییرات روندها

Processor Family Share of Top500

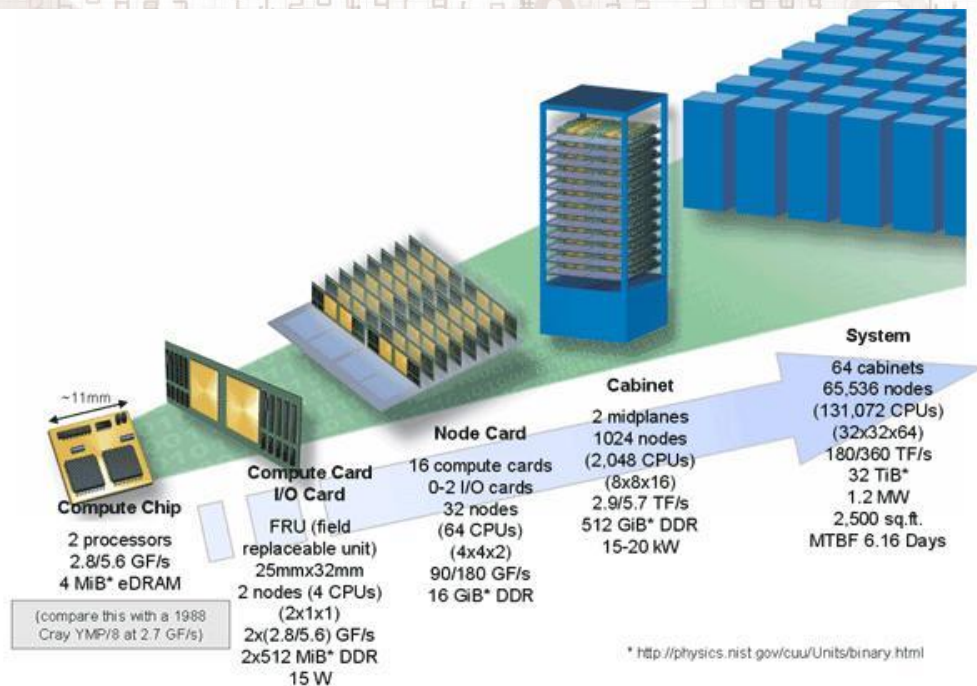


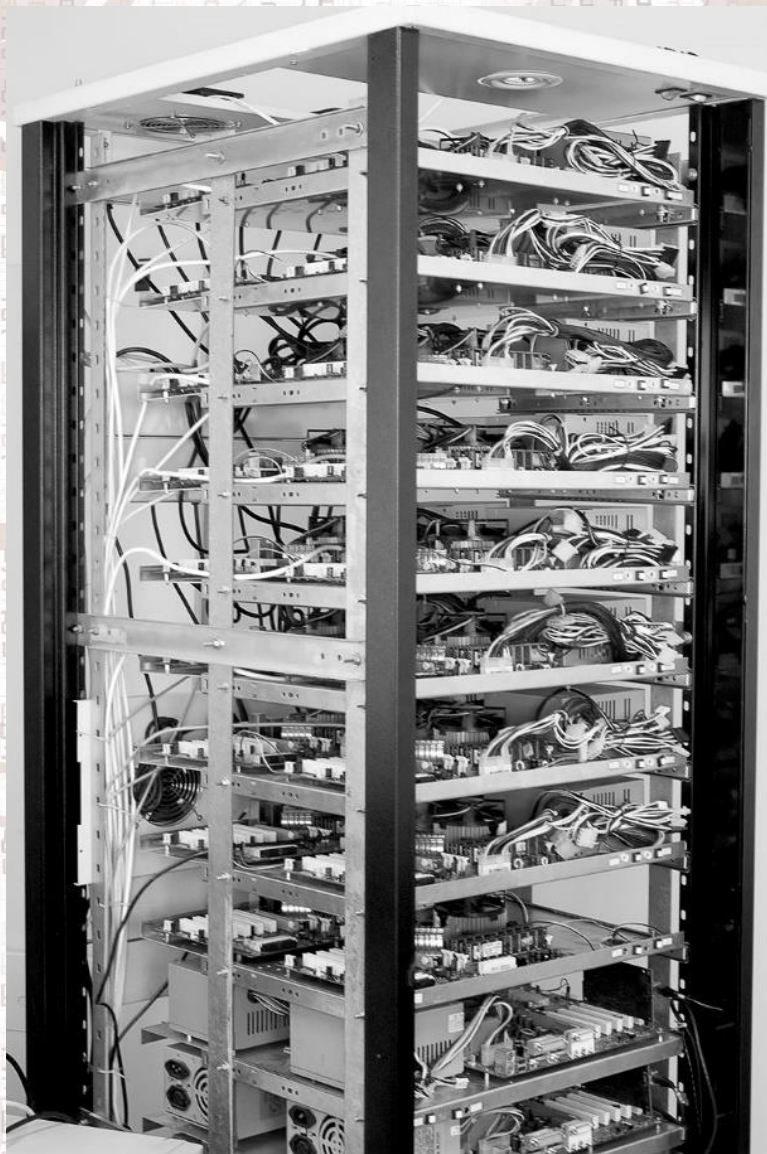
تغییرات روندها

Operating Systems Used On Top500 Supercomputers



Source: www.top500.org

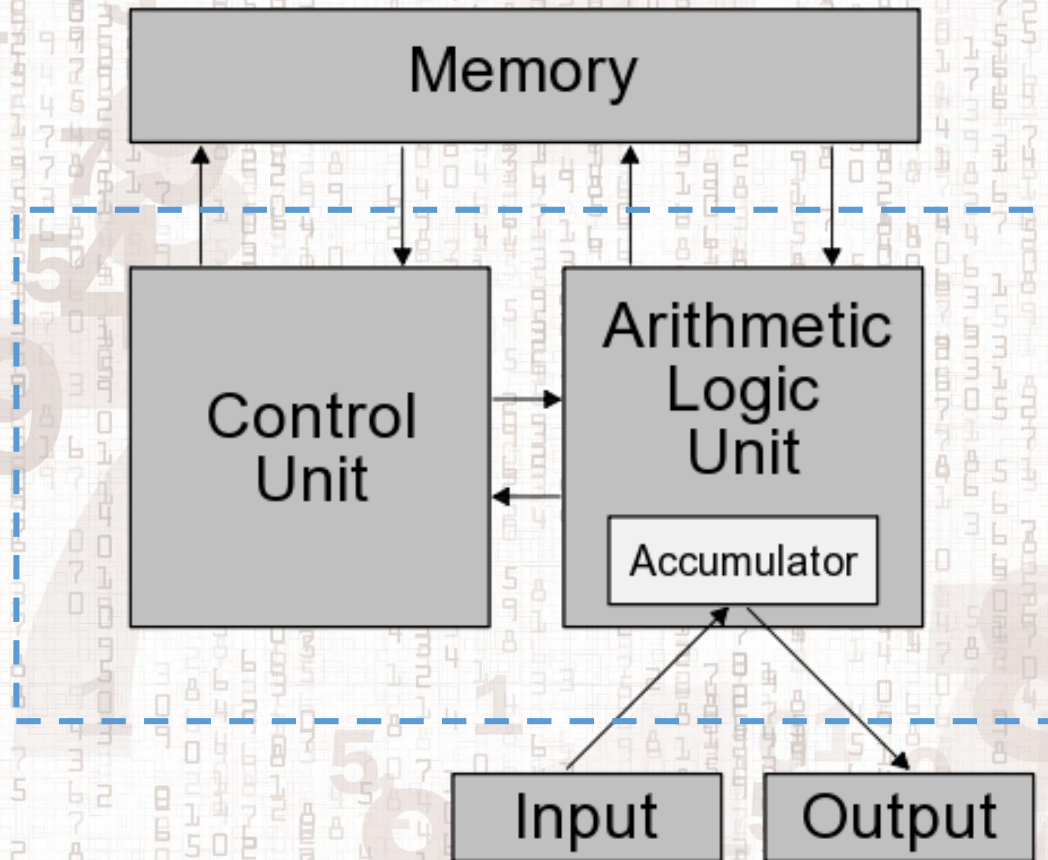




א

مدل فون نویمان

کامپیوتر با برنامه ذخیره شده



واحد پردازنده
(PU)

ارتباط پردازنده با حافظه

- امکان آرایش واحدهای پردازنده به شکل های مختلف

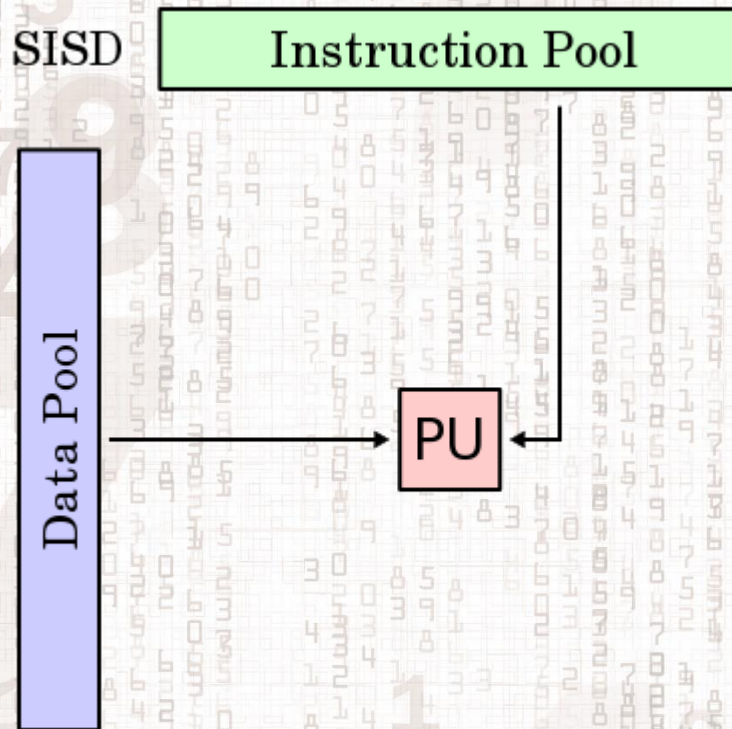
- دسته بندی سیستم های چند پردازنده ای بر حسب آرایش آنها توسط فلین در سال ۱۹۶۶

- دسته بندی بر مبنای مفهوم جریان

- جریان داده

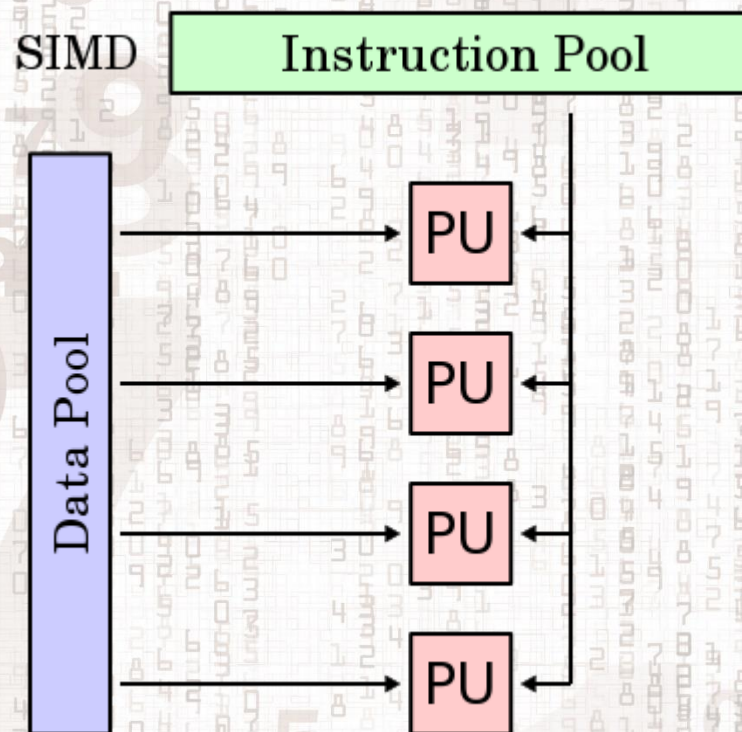
- جریان دستورالعمل

تک دستور-تک داده (SISD)



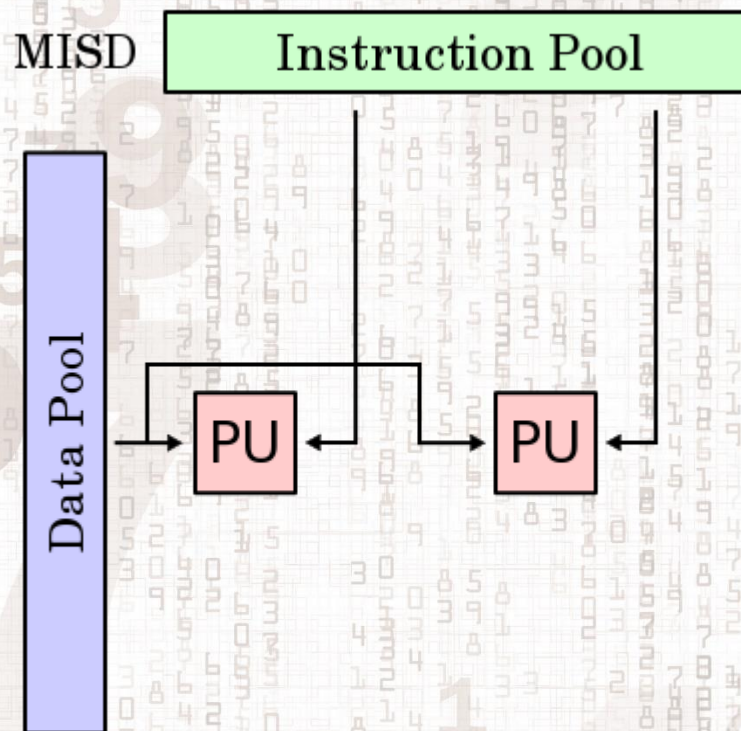
کامپیوتر کلاسیک فون نویمان

تک دستور-چند داده (SIMD)



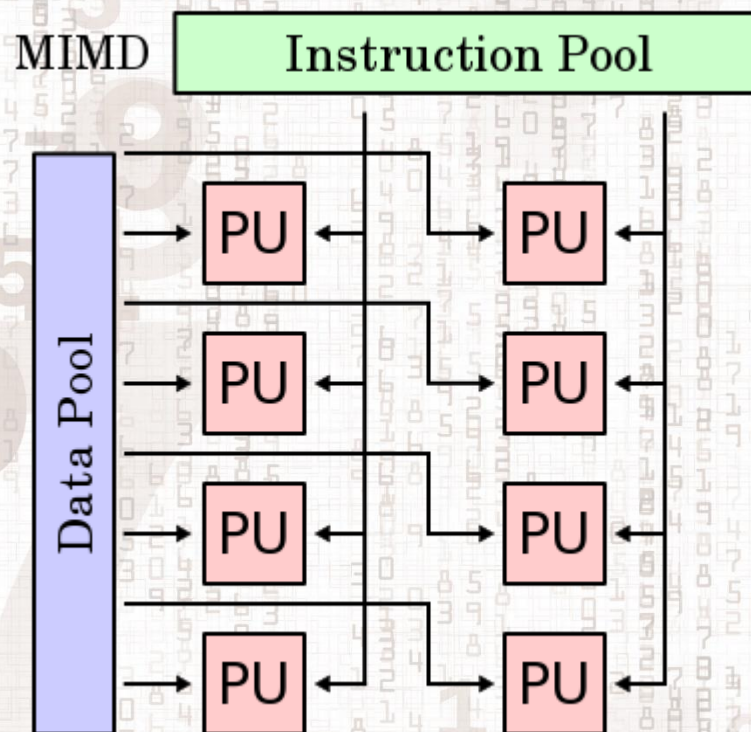
کامپیوترهای برداری

چند دستور-تک داده (MISD)



سیستم های مبتنی بر رای گیری

چند دستور-چند داده (MIMD)



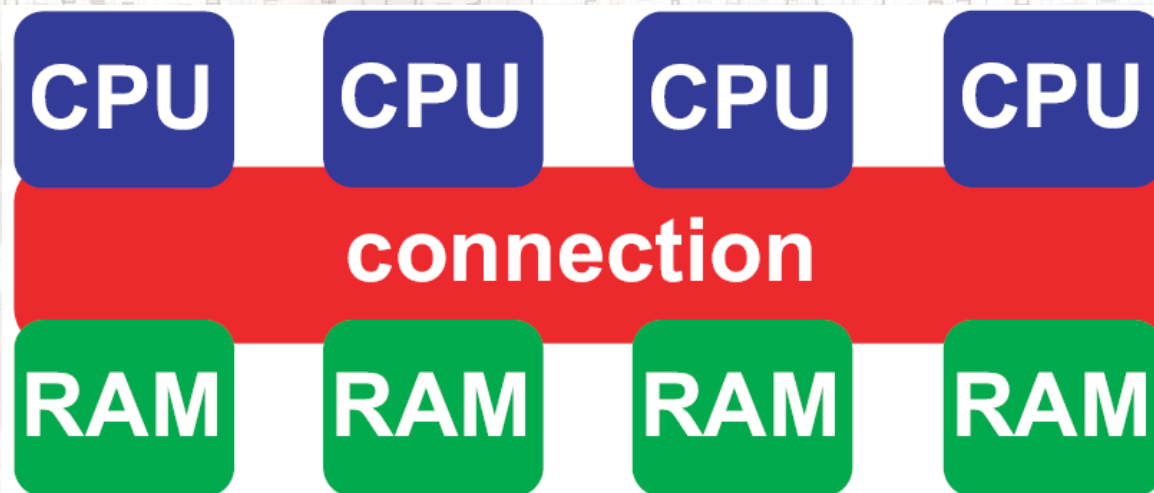
کامپیوترهای چند هسته/پردازنده عمومی

ارتباط پردازنده و حافظه

معماری MIMD بر اساس نحوه ارتباط پردازنده و حافظه به دو دسته عمده تقسیم می شود.

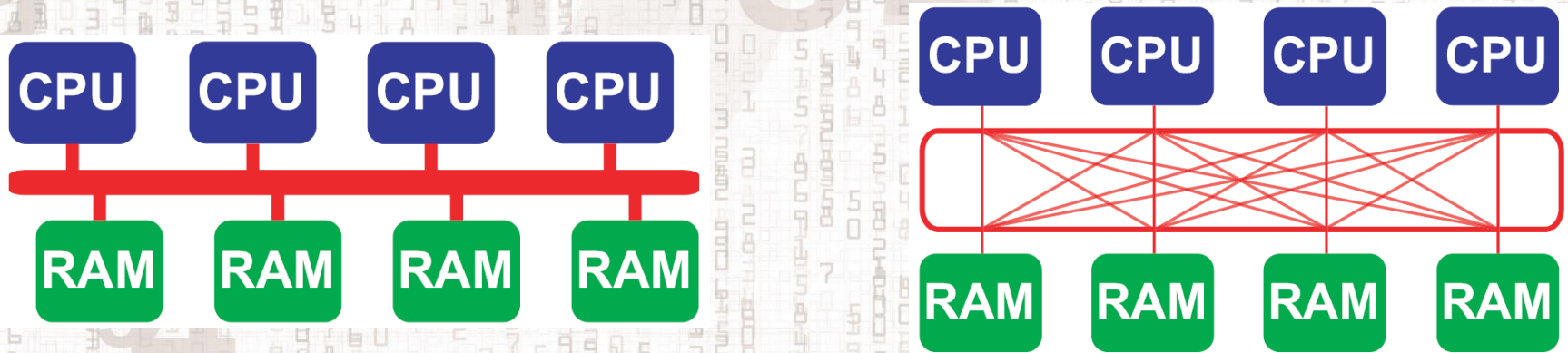
- حافظه مشترک (Shared Memory Architecture)
- دسترسی مستقیم هر پردازنده به تمامی حافظه
- حافظه گسترده (Distributed Memory Architecture)
- دسترسی مستقیم هر پردازنده تنها به حافظه محلی خود

معماری حافظه مشترک



معروف به (SMP) Symmetric Multi Processor

انواع سیستم های حافظه مشترک



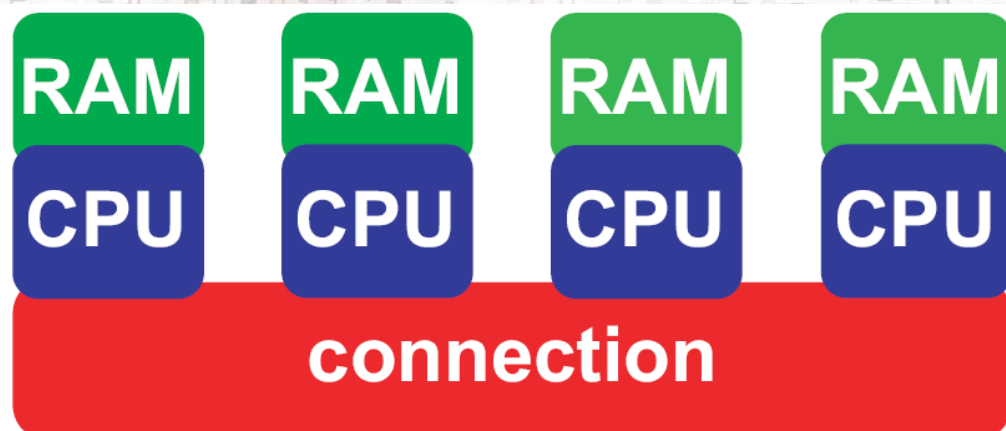
معماری گذرگاه (Bus)

- پیچیدگی طراحی $O(n)$
- پهنای باند $O(n^{-1})$
- زمان تاخیر $O(n)$

معماری سویچ (Cross bar)

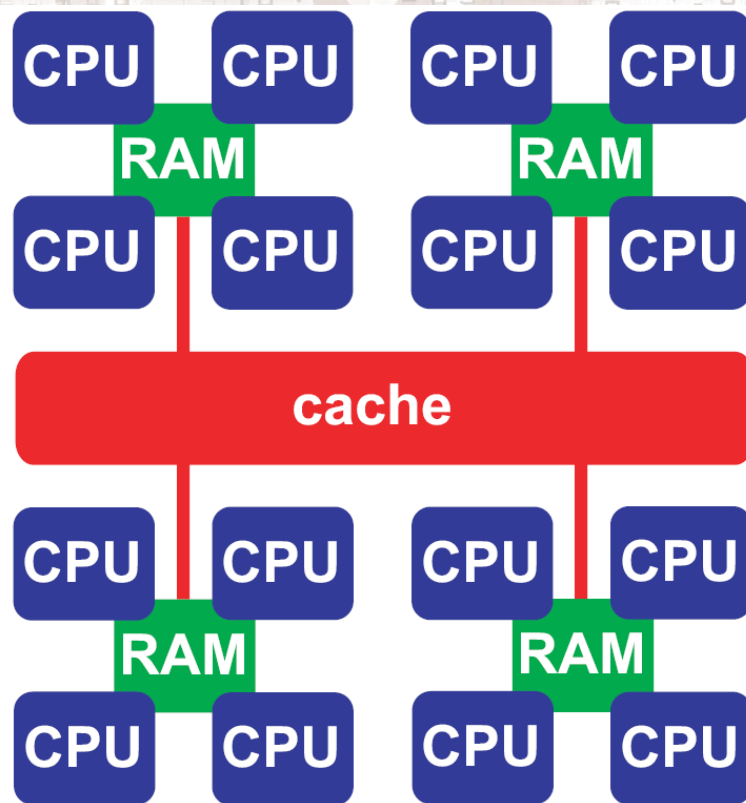
- پیچیدگی طراحی $O(n^2)$
- پهنای باند مستقل از n
- زمان تاخیر $O(1)$

معماری حافظه گسترده



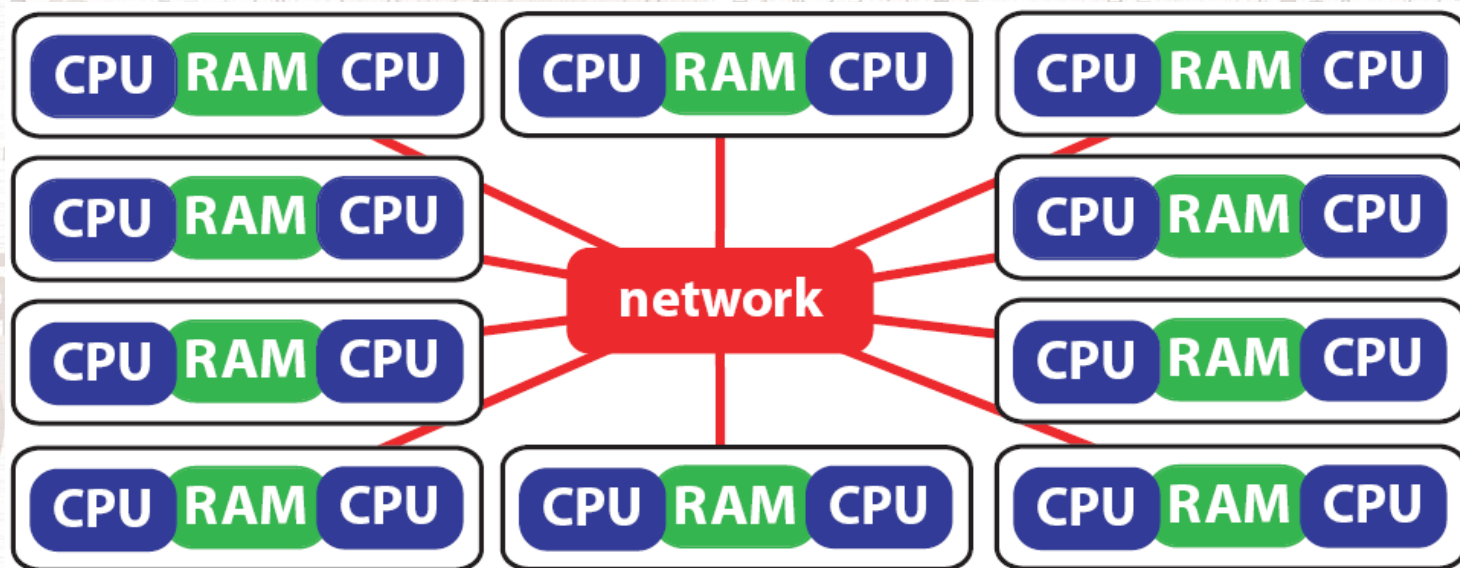
بسته به نوع ارتباط، طیف وسیعی از سیستم ها قابل طرح است

معماری حافظه گسترده



استفاده از حافظه کش به عنوان ابزار انتقال داده (CCNUMA)

معماری حافظه گسترده



- کلاستر: شبکه می تواند شکلی از LAN باشد
- تور: کلاستری که پردازنده های آن در محل های مختلف بوده و از طریق اینترنت متصل شده اند

ملاحظات نرم افزاری

برنامه به دو شکل کلی می تواند موازی شود

- موازی سازی داده ای (Data parallel)
- عملیات مشخص بر مجموعه های متفاوتی از داده ها به صورت همزمان اعمال خواهد شد
- موازی سازی عملیاتی (Task parallel)
- کارهای مختلف به صورت همزمان انجام می شود

روش موازی سازی

- تقسیم کار به خرد کارها
- شناسایی وابستگی میان خرد کارها
- تغییر ترتیب انجام کارها بگونه ای که کارهایی که وابستگی ندارند همزمان انجام شوند.

قابلیت موازی سازی کارها

کاملاً موازی شوند (Embarrassingly Parallel Tasks)

وابستگی بسیار کم میان کارها

- محاسبات اعداد تصادفی
- الگوریتم ژنتیک

قابلیت موازی سازی کارها

کارهای دانه درشت (Coarse Grained Tasks)

وجود وابستگی میان کارها، امکان خوشه بندی کارها در گروه های با
وابستگی کم
ضرورت انتقال داده ها

- روش های جستجو
- محاسبات پایه ای ماتریسی و جبر خطی

قابلیت موازی سازی کارها

کارهای دانه ریز (Fine Grained Tasks)

وجود وابستگی شدید میان کارها،

عدم امکان خوشه بندی کارها در گروه های با وابستگی کم
نیاز به انتقال داده زیاد

• روش های حل عددی معادلات دیفرانسیل با مشتق جزئی

ارزیابی عملکرد

زمان در برنامه های موازی

• زمان محاسبات

• زمان انتقال داده ها

زمان انتقال داده ها

پارامترهای مهم در مکانیزم ارتباطی

- پهنای باند (Bandwidth)
- حجم داده عبور داده شده در واحد زمان
- تاخیر (Latency)
- زمان لازم برای شروع انتقال داده

زمان انتقال داده ها

$$T_{comm} = T_l + \frac{\text{Data size}}{\text{Bandwidth}}$$

Medium	Bandwidth	latency
Cache	100 Gb/sec	1 cycle
RAM	10 Gb/sec	100 cycles
Fast Ethernet	100 Mb/sec	100 μ sec
Gigabit Ethernet	1000 Mb/sec	100 μ sec
Mayrinet	200 MB/sec	5 μ sec

ارزیابی عملکرد

عملکرد تئوری (Peak performance)

$$R_{peak} = n_{cores} n_{FPU} f$$

مثال

پنتیوم D 830

دو هسته

دو واحد محاسبه اعشاری در هر هسته

سرعت پردازنده 3GHz

$$R_{peak} = 2 \times 2 \times 3 = 12 \text{ GFLOPS}$$

ارزیابی عملکرد

عملکرد واقعی (Actual performance)

$$R = \frac{n_{FLOP}}{T_{Wall}}$$

راندمان محاسباتی

$$\epsilon = \frac{R}{R_{peak}}$$

افزایش سرعت

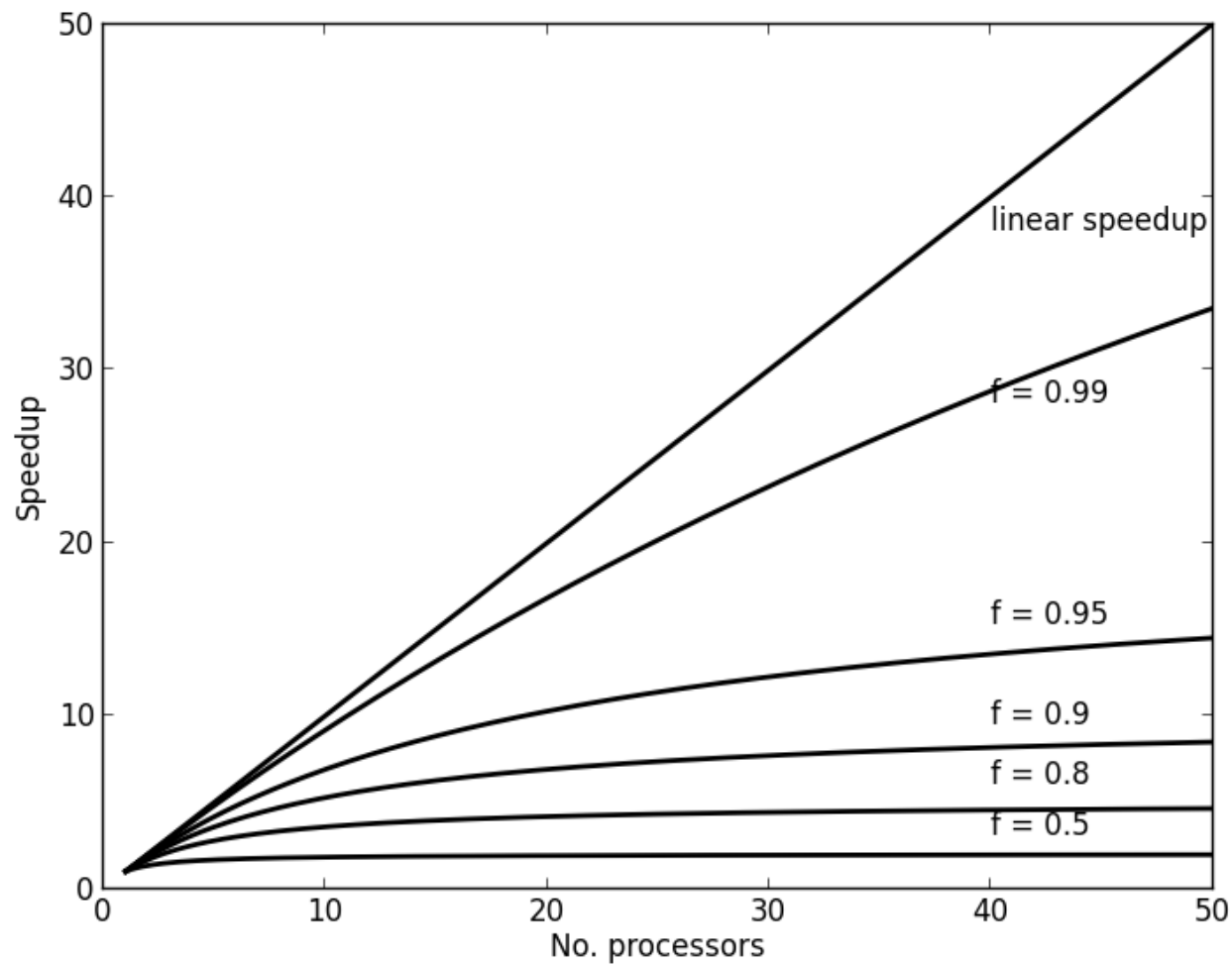
Speed up

$$S = \frac{T_{serial}}{T_{parallel}}$$

قانون امدال (Amdahl's law)

$$S = \frac{1}{1 - F + \frac{F}{S_p}}$$

افزایش سرعت



مقیاس پذیری

قابلیت حفظ عملکرد برنامه موازی با افزایش تعداد پردازنده ها

- مقیاس پذیری قوی (Strong scaling)

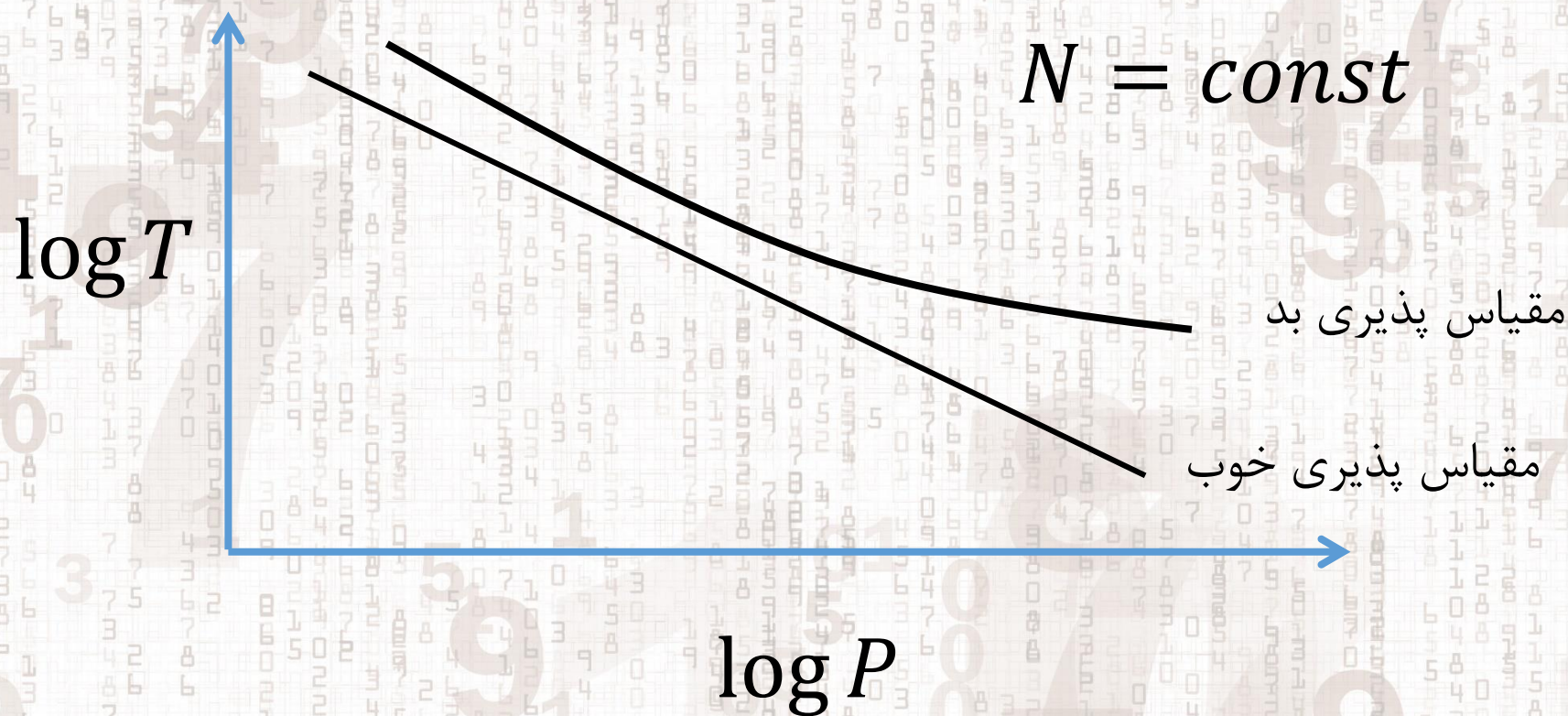
مقیاس پذیری امدال

- مقیاس پذیری ضعیف (Weak scaling)

مقیاس پذیری گوستافسون

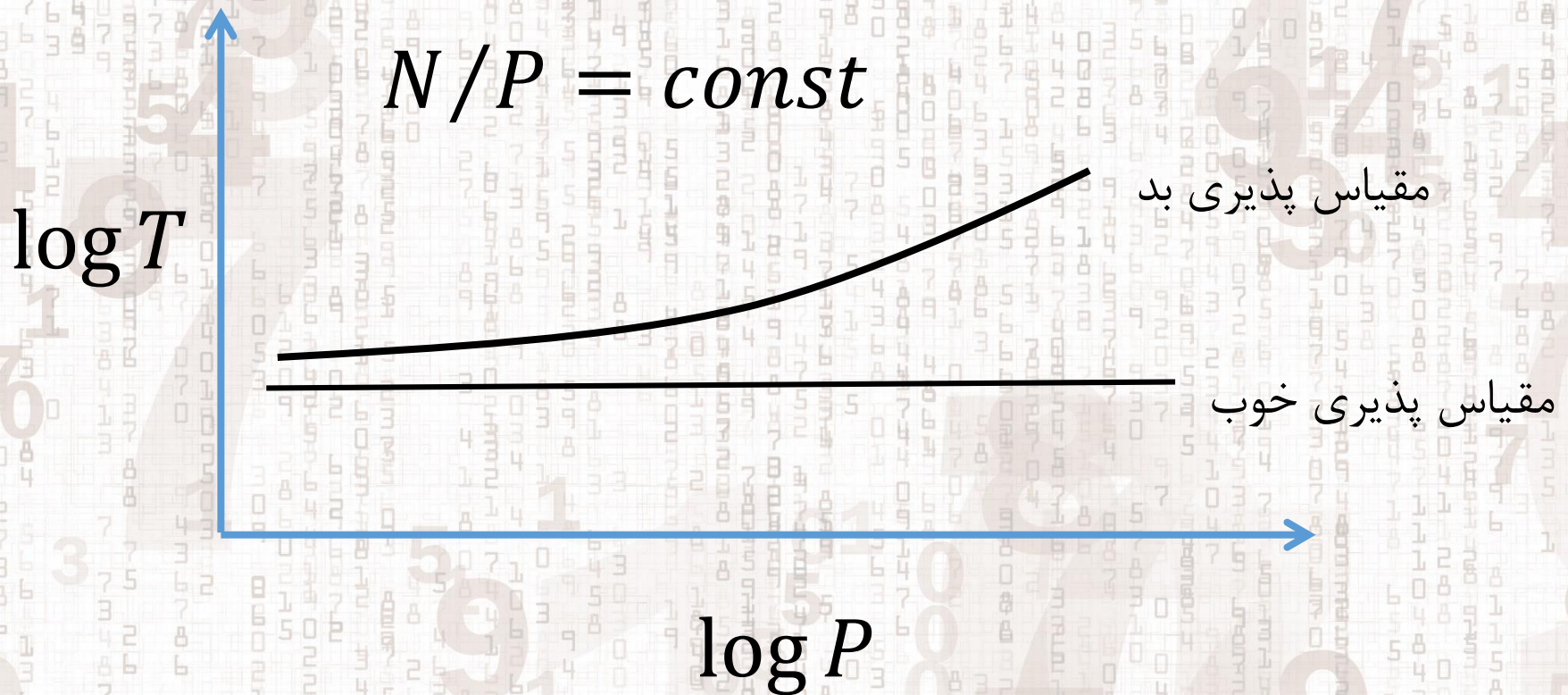
مقیاس پذیری قوی

کاهش زمان اجرا برای حجم کار مشخص با افزایش تعداد پردازنده ها



مقیاس پذیری ضعیف

ثابت ماندن زمان اجرا با افزایش تعداد پردازنده ها به ازای حجم کار مشخص در هر پردازنده



اهداف موازی سازی

در موازی سازی برنامه ها سه هدف زیر دنبال می شود

- تطابق پاسخ ها با برنامه سری

- سرعت افزایی

- مقیاس پذیری

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

الحمد لله

الحمد لله الذي هدانا لهذا الذي كنا لنهتدي لولا أن هدانا الله

ابزارهای موازی سازی

برنامه نویسی برای حافظه مشترک

- مزایا
 - پیچیدگی کمتر در دسترسی به حافظه
 - پشتیبانی سیستم عامل
 - در دسترس بودن سخت افزار
- معایب
 - محدودیت گسترش پذیری حافظه
 - محدودیت گسترش پذیری پردازنده
- ابزار برنامه نویسی
 - OpenMP

ابزارهای موازی سازی

برنامه نویسی برای حافظه گسترده

- مزایا

- توسعه پذیری خوب حافظه

- توسعه پذیری خوب پردازنده

- معایب

- پیچیدگی زیاد در دسترسی به حافظه

- لزوم طراحی مجدد و اصلاح الگوریتم ها

- پیچیدگی برنامه نویسی و پیاده سازی

- ابزار برنامه نویسی

- PVM - MPI

معرفی OpenMP

- مجموعه دستورات اضافه به کامپایلر

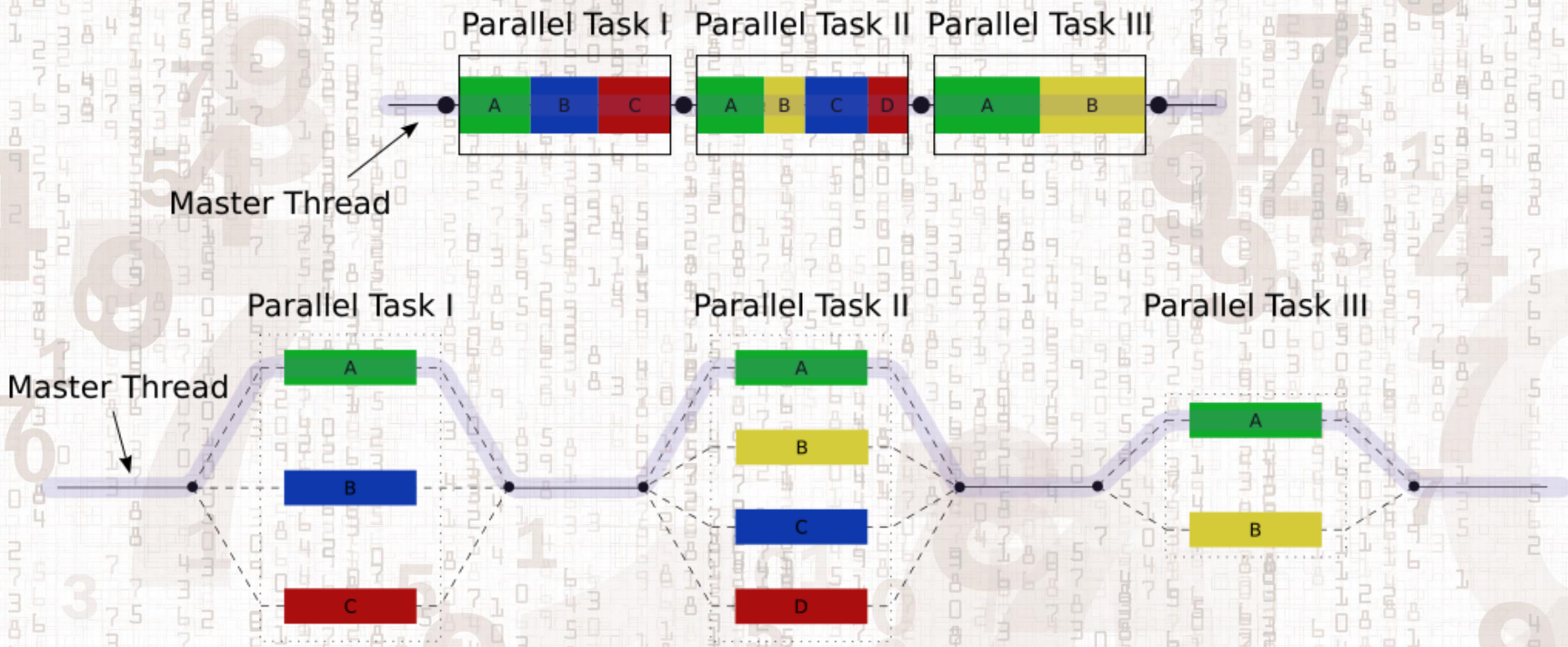
- Fortran

- C/C++

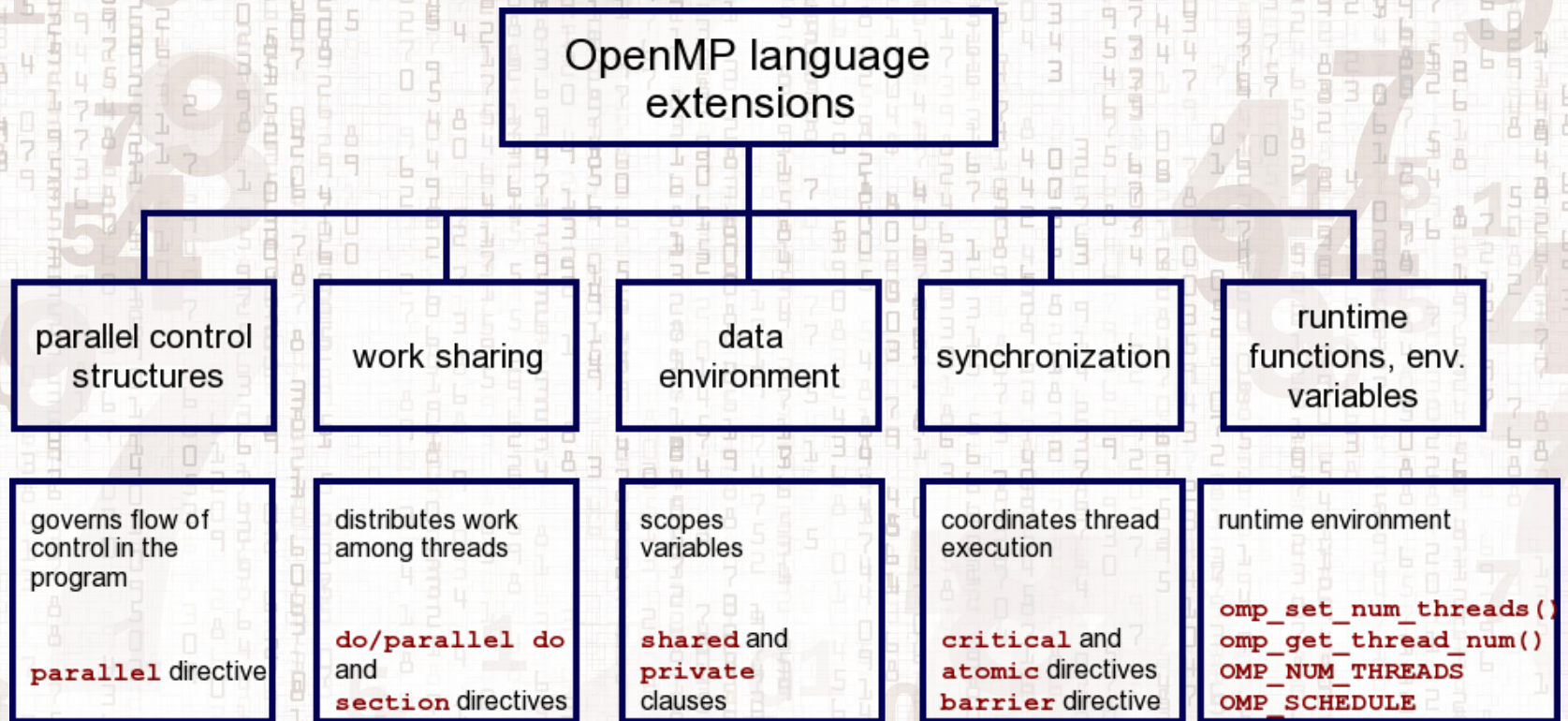
- استفاده از ریسمان ها (Threads)

- هدایت کامپایلر برای ایجاد کد اضافه برای موازی سازی

ملا اجرای OpenMP



اجزای OpenMP



ایجاد ناحیه موازی

```
!!$ omp parallel  
    code block  
!!$ omp end parallel
```

Fortran

```
!!  
#pragma omp parallel  
    code block
```

C/C++

ایجاد ناحیه موازی

```
#include <stdio>
#include <omp.h>

int main() {
    int n = 4, i;
    #pragma omp parallel num_threads( n ) private( i )
    {
        i = omp_get_thread_num();
        printf( "Hello from thread No. %d\n", i );
    }
    return 0;
}
```

ایجاد ناحیه موازی

```
use omp_lib
implicit none
integer n, i

!$omp parallel num_threads( n ) private( i )
  i = omp_get_thread_num()
  write( *, * ) 'Hello from thread ', I
!omp end parallel
```

ایجاد ناحیه موازی

```
Hello from thread No. 0  
Hello from thread No. 2  
Hello from thread No. 1  
Hello from thread No. 3
```

دسترسی به متغیرها

دسترسی به متغیرها در برنامه به دو صورت قابل تعریف است

- مشترک (shared)
 - همه ریسمانها به متغیر دسترسی دارند
 - تغییر ایجاد شده در متغیر توسط یک ریسمان، در دیگر ریسمان ها دیده می شود
- اختصاصی (private)
 - هر ریسمان نسخه اختصاصی از متغیر را در اختیار دارد
 - تغییر ایجاد شده در متغیر توسط یک ریسمان، تاثیری بر مقدار متغیر در دیگر ریسمان ها ندارد

مسایل مربوط به متغیرها

- دسترسی همزمان به متغیر مشترک باعث ایجاد خطای منطقی می شود
- شرایط مسابقه ای (Data race condition)
- دسترسی به صورت فقط خواندنی اشکال ندارد
- این شرایط شکلی از وابستگی داده ای است
- برای موازی سازی باید وابستگی میان ریسمان ها از بین برود

تقسیم کار

با کمک OpenMP میتوان کار را میان ریسمان ها تقسیم کرد

- تقسیم کار در حلقه
- تقسیم تکرارهای حلقه میان ریسمان ها
- تقسیم کارهای مستقل
- انجام کارهای متفاوت توسط ریسمان های مختلف
- تقسیم کار باید حتماً درون `omp parallel` باشد

موازی سازی حلقه ها

Fortran

```
!$ omp do  
do loop  
!$ omp end parallel
```

C/C++

```
#pragma omp for  
for loop
```

تقسیم کار در حلقه

```
#include <stdio>
#include <omp.h>

int main(){
    int m, n, i, j;
    float a[ m, n ], b[ n ], c[ m ];
    #pragma omp parallel private( i, j )
        #pragma omp for
        for( i = 0, i < m; i++ )
            c[ i ] = 0;
            for( j = 0; j < n; j++ )
                c[ i ] = a[ i, j ] * b[ j ];
    return 0;
}
```

تقسیم کار در حلقه

```
use omp_lib
implicit none
integer n, m, i, j
real a( m, n ), b( n ), c( m )
!$omp parallel private( i, j )
!$omp do
  do i = 1, m
    c( i ) = 0
    do j = 1, n
      c( i ) = a( i, j ) * b( j )
    end do
  end do
!$omp end do
!omp end parallel
```

تقسیم کارهای مستقل

Fortran

```
...  
!$ omp sections  
  [!$omp section]  
    code block  
  [!$omp section]  
    code block  
  ...  
!$ omp end sections  
...
```

C/C++

```
...  
#pragma omp sections  
{  
  [#pragma omp section]  
    code block  
  [#pragma omp section]  
    code block  
  ...  
}  
...
```

استفاده از OpenMP

- مزایا
 - سادگی استفاده
 - امکان موازی سازی تدریجی
 - عدم نیاز به تغییرات عمده در کد
- معایب
 - مقیاس پذیری محدود
 - اختصاص داشتن به ماشین های حافظه مشترک

معرفی MPI

- استاندارد و کتابخانه های لازم برای انتقال پیام
- Message Passing Interface

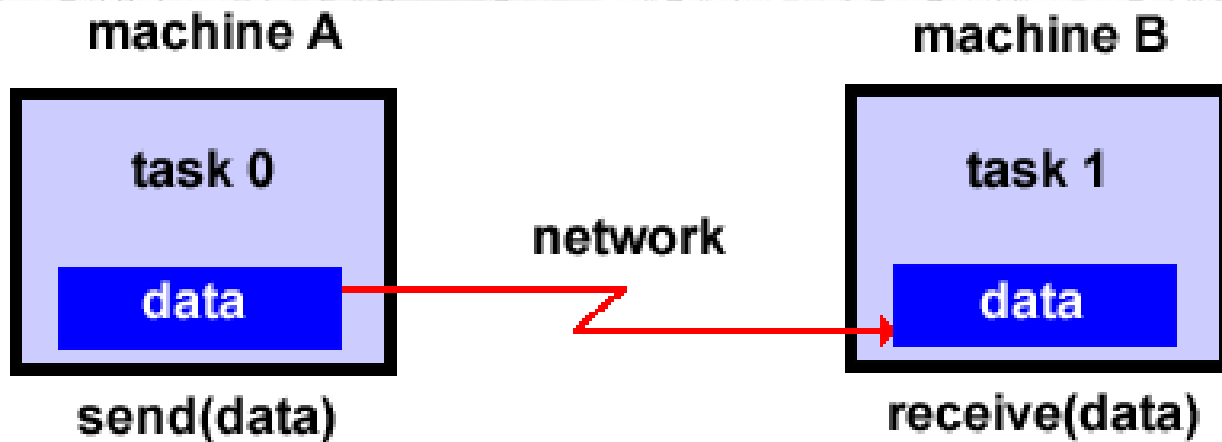
- قابلیت بکارگیری در زبان های برنامه نویسی
- Fortran
- C/C++

- قابلیت استفاده در ماشین های حافظه گسترده و مشترک

ویژگی های برنامه نویسی با MPI

- برنامه در تعدادی پروسس اجرا می شود
- هر پروسس یک نسخه از کد را به صورت مستقل اجرا می کند
- هر پروسس فقط به حافظه اختصاصی خود دسترسی دارد
- تبادل داده میان پروسس ها باید به صورت صریح توسط برنامه نویس انجام شود.
- انتقال داده ها توسط سخت افزار رابط انجام می شود
- عملکرد برنامه به شدت تابع ویژگی های رابط انتقال است

مدل تبادل پیام



انتقال پیام به صورت نقطه به نقطه
Point to Point communication (P2P)

برنامه ساده MPI

```
#include <iostream>
#include <mpi.h>

int main( argc, char *argv[] ){
    int myRank, size;
    MPI_Init( argc, argv );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );

    printf( "Hello, I am process \", rank, " of \", size );

    MPI_Finalize();
    return 0;
}
```

برنامه ساده MPI

PROGRAM MAIN

USE MPI

INCLUDE "mpif.h"

INTEGER MYRANK, MYSIZE, IERR

call MPI_INIT(IERR)

call MPI_COMM_RANK(MPI_COMM_WORLD, MYRANK, IERR)

call MPI_COMM_SIZE(MPI_COMM_WORLD, MYSIZE, IERR)

PRINT *, 'Hello, I am process ', MYRANK, ' of ', MYSIZE

call MPI_FINALIZE(IERR)

STOP

END

انتقال پیام P2P

هر ارتباط P2P دو بخش دارد

- فرستنده پیام
- ارسال توسط MPI_Send
- گیرنده پیام
- دریافت توسط MPI_Recv

انتقال پیام P2P

```
char msg[20];
int myrank, tag = 99;
MPI_Status status;
...
MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
if( myrank == 0 ){
    strcpy( msg, "Hello there");
    MPI_Send( msg, strlen(msg)+1, MPI_CHAR, 1,
              tag, MPI_COMM_WORLD);
}
else if( myrank == 1 ){
    MPI_Recv( msg, 20, MPI_CHAR, 0,
              tag, MPI_COMM_WORLD, &status);
}
```

انتقال پیام P2P

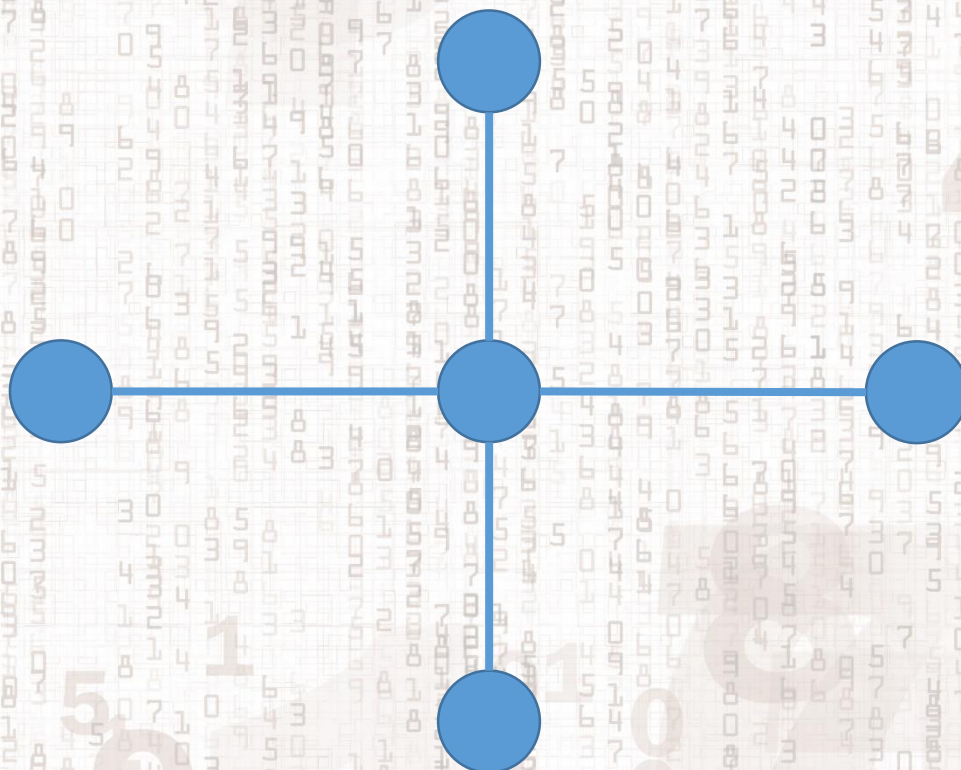
```
CHARACTER*20 msg
INTEGER myrank, ierr, status(MPI_STATUS_SIZE)
INTEGER tag = 99
...
call MPI_COMM_RANK( MPI_COMM_WORLD, myrank, ierr )
IF (myrank .EQ. 0) THEN
    msg = "Hello there"
    call MPI_SEND( msg, 11, MPI_CHARACTER, 1,
                  tag, MPI_COMM_WORLD, ierr)
ELSE IF (myrank .EQ. 1) THEN
    call MPI_RECV( msg, 20, MPI_CHARACTER, 0,
                  tag, MPI_COMM_WORLD, status, ierr)
END IF
```

استراتژی موازی سازی با MPI

استراتژی تقسیم دامنه (Domain Decomposition)

- تقسیم دامنه محاسباتی به بخش های کوچکتر
- انجام محاسبات در زیر دامنه ها
- تبادل اطلاعات روی مرزها

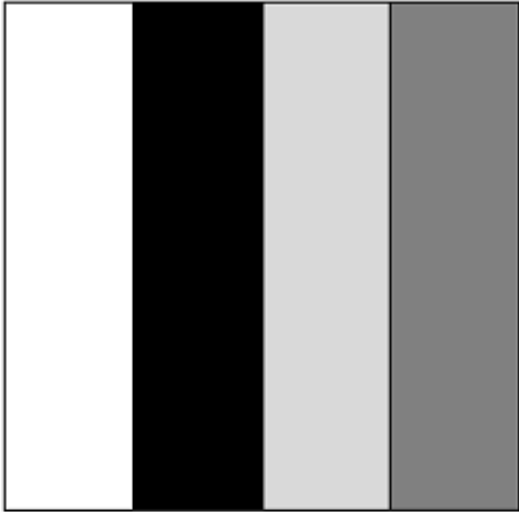
روش هموار سازی ژاکوبی



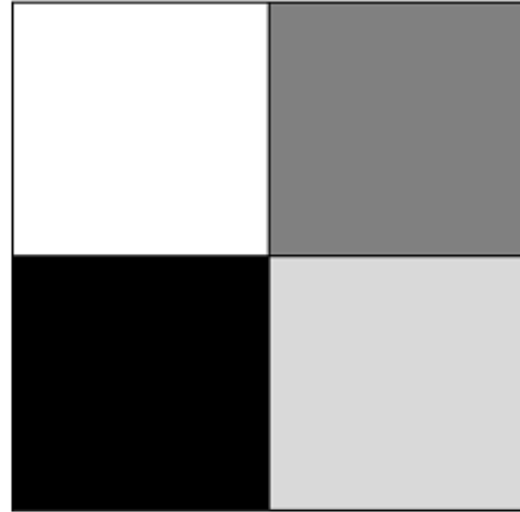
روش هموار سازی ژاکوبی

```
1 real, allocatable A(:, :), B(:, :)  
2 allocate(A(0:n+1, 0:n+1), B(n,n))  
3  
4 do while (.NOT. converged)  
5     do j = 1,n  
6         do i = 1,n  
7             B(i,j) = 0.25*(A(i-1,j)+A(i+1,j)  
8                 +A(i,j-1)+A(i,j+1))  
9         end do  
10    end do  
11    do j = 1,n  
12        do i = 1,n  
13            A(i,j) = B(i,j)  
14        end do  
15    end do  
16 end do
```

تقسیم دامنه

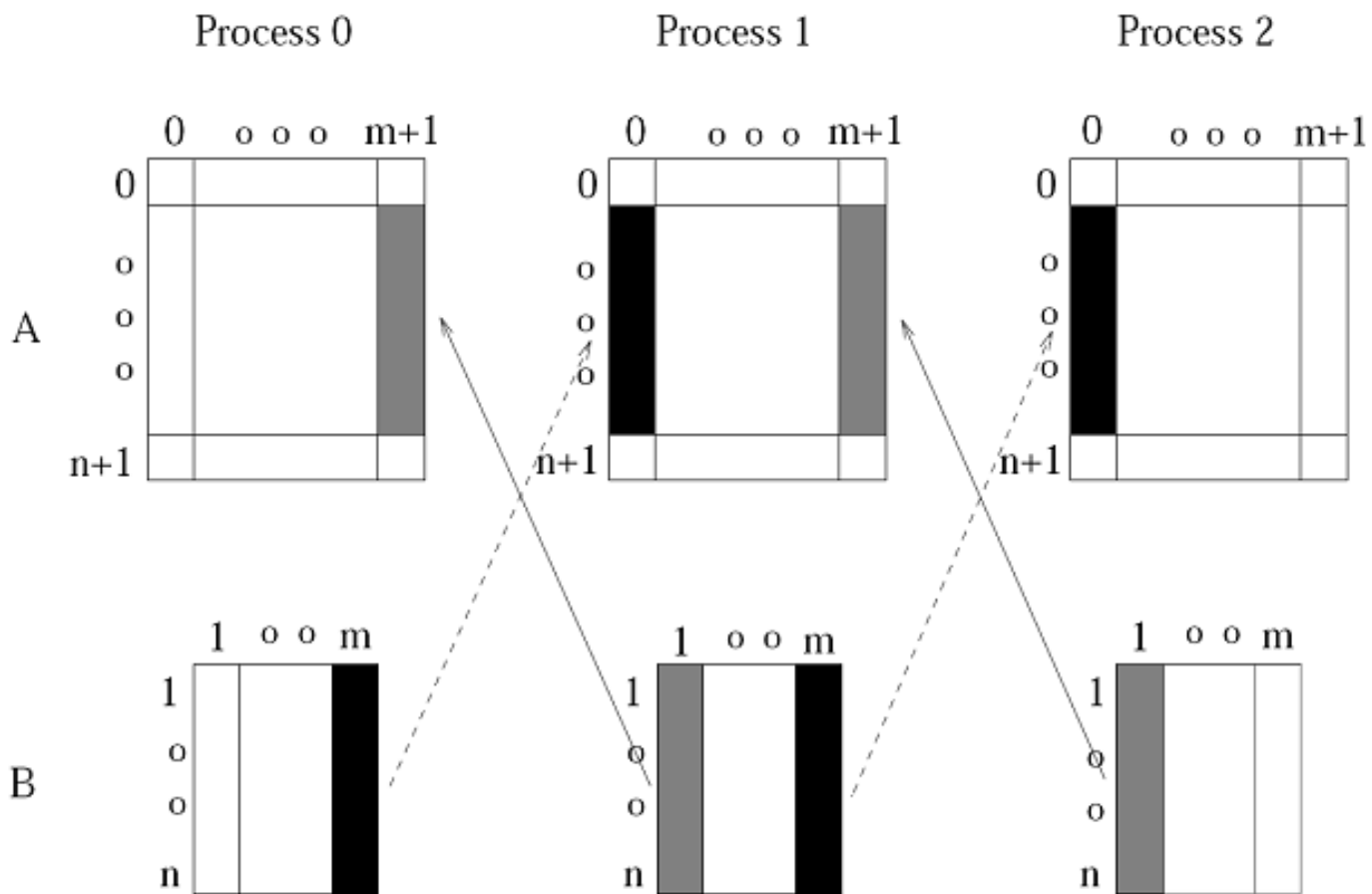


1D partition



2D partition

نحوه تبادل داده ها



تقسیم دامنه

```
1 real, allocatable A(:, :), B(:, :)  
2 ...  
3  
4 ! Get the number of processes and the rank of this process  
5 call MPI_COMM_SIZE(comm, p, ierr)  
6 call MPI_COMM_RANK(comm, myrank, ierr)  
7  
8 ! Compute size of local block  
9 m = n/p  
10 if (myrank .LT. (n-p*m)) then  
11     m = m+1  
12 end if  
13  
14 ! Allocate local arrays  
15 allocate (A(0:n+1, 0:m+1), B(n, m))
```

انجام محاسبات در زیر دامنه

```
1 ! Iterate in the block
2 do while (.NOT. converged)
3     do j = 1,m
4         do i = 1,n
5             B(i,j) = 0.25*(A(i-1,j)+A(i+1,j)
6                 +A(i,j-1)+A(i,j+1))
7         end do
8     end do
9     do j = 1,m
10        do i = 1,n
11            A(i,j) = B(i,j)
12        end do
13    end do
14 end do
```

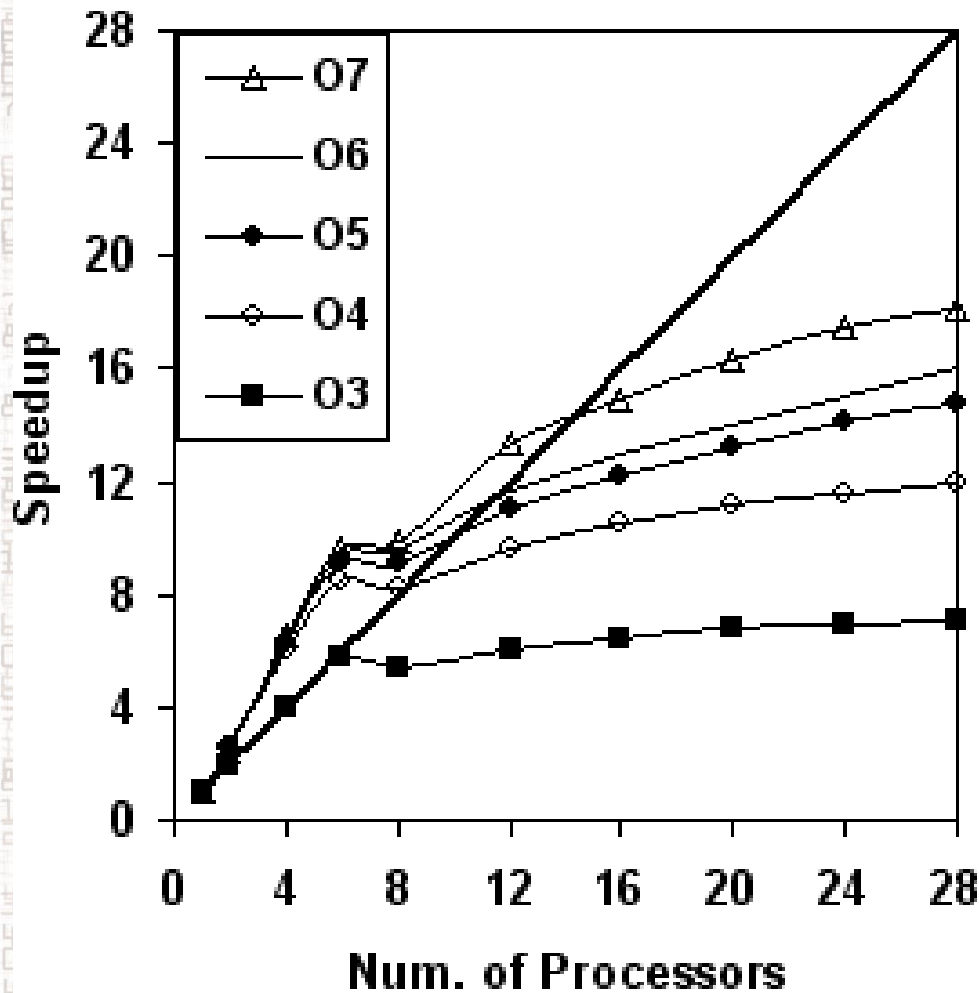
تبادل داده ها

```
1 ! Communicate to exchange data
2 if (myrank .GT. 0) then
3     call MPI_SEND(B(1,1), n, MPI_REAL, myrank-1, tag, comm,
4                   ierr)
5     call MPI_RECV(A(1,0), n, MPI_REAL, myrank-1, tag, comm,
6                   st, ierr)
7 end if
8 if (myrank .LT. p-1) then
9     call MPI_SEND(B(1,m), n, MPI_REAL, myrank+1, tag, comm,
10                  ierr)
11     call MPI_RECV(A(1,m+1), n, MPI_REAL, myrank+1, tag, comm,
12                  st, ierr)
13 end if
```

نکات تکمیلی

- تکنیک های پیشرفته موازی سازی
 - تبادل داده جمعی
 - پنهان سازی تبادل داده ها
 - استفاده از کش
- افزایش سرعت فوق خطی Superlinear speedup
 - استفاده از GPU
 - استفاده از FPGA

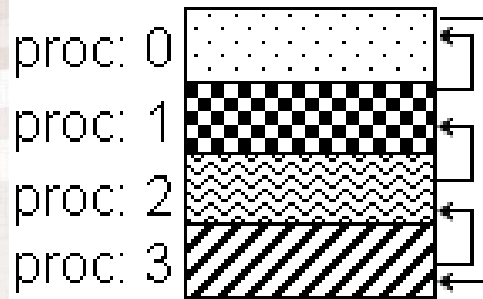
مطالعه موردی معکوس سازی ماتریس



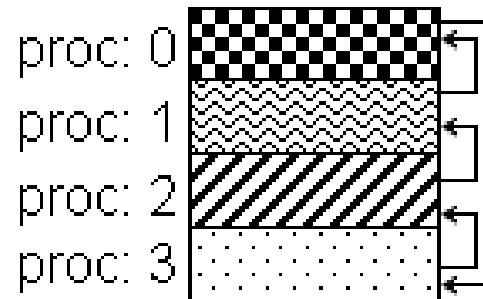
مطالعه موردی

معکوس سازی ماتریس

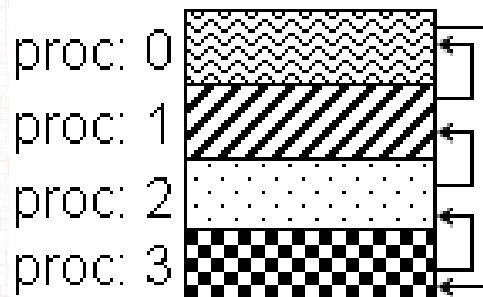
Step 1



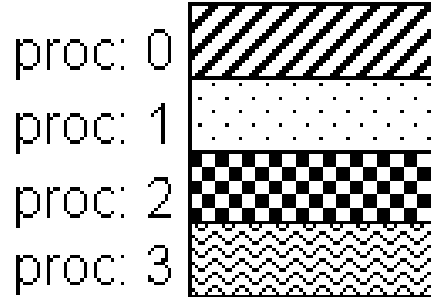
Step 2



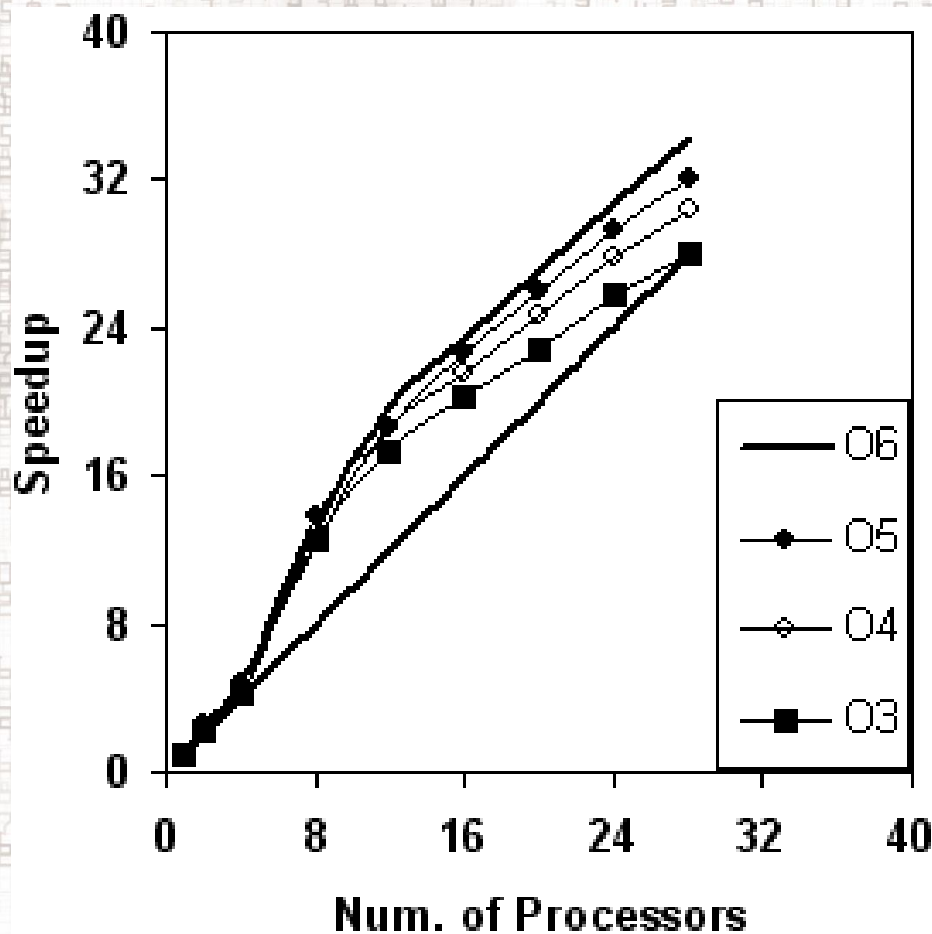
Step 3



Step 4



مطالعه موردی معکوس سازی ماتریس



جمع بندی

- معرفی پردازش موازی
- ویژگی های مهم در پردازش موازی
 - جنبه های سخت افزاری
 - جنبه های نرم افزاری
- معرفی ابزارهای برنامه نویسی موازی
 - OpenMP
 - MPI

کارگاه آموزشی

- برنامه نویسی OpenMP

- آموزش OpenMP

- ساختارهای پیشرفته تقسیم کار

- بهبود عملکرد و رفع گلوگاه ها

- برنامه نویسی MPI

- تبادل پیام نقطه به نقطه

- تبادل پیام غیر انسدادی

- تبادل پیام جمعی

- منابع بروز خطا

باتشکر

برخی منابع مفید

- Karniadakis, G.E., Kirby II, R.M., 2003, **Parallel Computing in C++ and MPI**, Cambridge University Press.
- Snir, M., Gropp, W., 1998, **MPI: The Complete Reference**, MIT Press, MA., USA.
- Chandra, R., et.al., 2001, **Parallel Processing using OpenMP**, Academic Press, CA, USA.
- Bertsekas, D.P. and Tsitsiklis, J.N., 1997, **Parallel and Distributed Computation: Numerical Methods**, Athena Scientific, Belmont, MA., USA.
- Bruaset, A.M. and Tveito, A., (Eds.), 2006, **Numerical Solution of Partial Differential Equations on Parallel Computers**, Springer-Verlag, Berlin, Germany.
- Turner, P.R. (Ed.), 1989, **Numerical Analysis and Parallel Processing**, Lecture Notes in Mathematics, Dold, A. and Eckmann, B. (Eds.), Springer-Verlag, Berlin, Germany.
- El-Rewini, H., Abd El-Barr, M., 2005, **Advanced Computer Architecture and Parallel Processing**, Wiley Interscience, Hoboken, NJ., USA.
- Wittwer, T., 2006, **An Introduction to Parallel Programming**, VSSD publication, Delft, The Netherlands.