به نام خدا

جزوه درسی

پردازش موازی

(Parallel Processing)

استاد : آقای دکترعلی برومند نیا

تهیه شده توسط :

مهرداد برزگر شبستری

شماره دانشجویی : ۸۸۰۳۲۴۱۶۰

دانشگاه پیام نور استان تهران

ترم دوم سال تحصیلی ۸۹-۹۰

مقدمه

کتاب مرجع :

Introduction to Parallel Processing

Algorithms and Architecture

Behrooz Parhami
University of California at Santa Barbara
Santa Barbara, California

نویسنده : بهروز پرهامی

محتویات

این کتاب از ۶ قسمت تشکیل شده است که هر قسمت دارای فصلهای مختلفی می باشد .

قسمت اول (مفاهیم اساسی):

قسمت دوم (مدلهای اضافی):

۵. الگوریتمهای مبنایی برای PRAM (Parallel Random Access Machine)

۵.۱.زیر مدلهای PRAM و فرضیات

۵.۲.انتشار داده ها

۵.۳.محاسبات Semigroup

۵.۴.محاسبات Parallel Perfix

۵.۵.رتبه بندی عناصر یک لیست پیوندی

۵.۶.ضرب ماتریسها

۶. الگوریتمهای بیشتر برای معماری حافظه اشتراکی (Shared-Memory)

۶.۱. انتخاب مبتنی برپایه رتبه ترتیبی

۶.۲. الگوریتم موازی برای انتخاب

۶.۳. الگوریتم مرتب سازی بر مبنای انتخاب

۶.۴. الگوریتمهای جایگزین برای مرتب سازی

۶.۵. پوسته برجسته از مجموعه نقاط ۲ بعدی

۶.۶. برخی از جنبه های پیاده سازی

۷. شبکه های انتخاب و مرتب سازی

۷.۱. شبکه مرتب سازی چیست ؟

۷.۲. نمایشی از شایستگی برای شبکه مرتب سازی

۷.۳. طراحی شبکه های مرتب سازی

۷.۴. شبکه مرتب سازی به روش دسته بندی (Batcher)

۷.۵. کلاسهای دیگری از شبکه های مرتب سازی

۷.۶. شبکه های انتخاب

۸. مثالهایی برای دیگر سطوح چرخشی

۸.۱. عملیات جستجو در فرهنگ لغات

۸.۲. ساختار درختی ماشین دیکشنری

۸.۳. محاسبات Parallel Perfix

۸.۴. شبکه های Parallel Perfix

۸.۵. تبدیل فوریه گسسته (DFT)

۸.۶. معماری موازی برای FFT

قسمت سوم (معماریهای بر مبنای مش):

۹. مرتب سازی با شبکه های مش ۲ بعدی (2D Mesh)

۹.۱.کامپیوترهای متصل شده بصورت مش

۹.۲.الگوریتم مرتب سازی برشی (Shearsort)

۹.۳.انواع ساده Shearsort

۹.۴.الگوریتمهای مرتب سازی بازگشتی

قسمت اول (مفاهیم اساسی)

شاخه پردازش موازی با معماری و روشهای مختلف الگوریتم نویسی به منظور افزایش عملکرد و یا ویژه گیهایی دیگر نظیر (مقرون به صرفه بودن و قابلیت اطمینان) مربوط می باشد.

اهداف پردازش موازی :

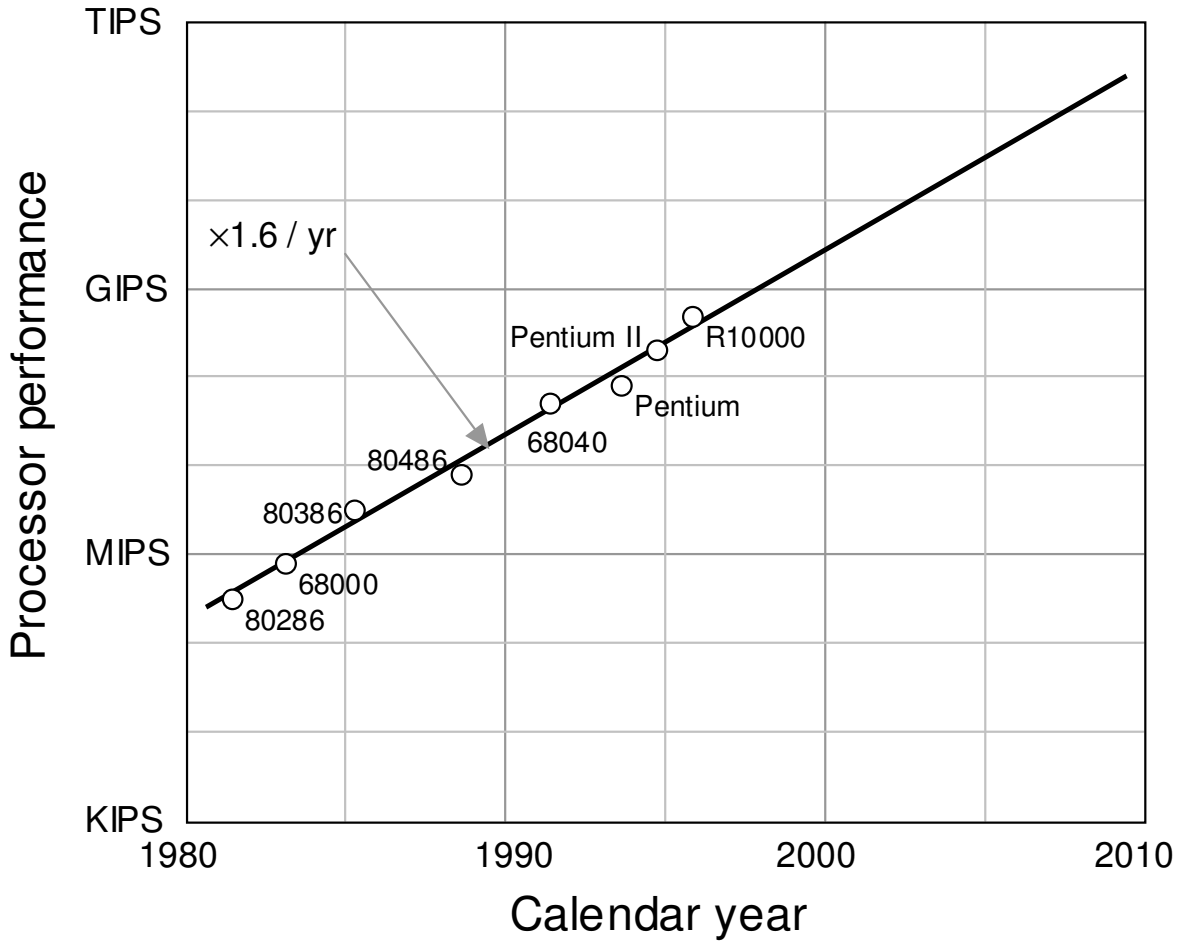
۱. **افزایش سرعت :** یعنی مسایل سریعتر با کامپیوتر حل شوند . مثل کنترل خطوط هوایی (Hard) یا دیدن ویدئو روی کامپیوتر (Soft)
۲. **گذردهی بالاتر :** یعنی کامپیوترهای مثل هم بتوانند تعداد زیادی مسائل را به صورت همزمان حل کنند.
۳. **قدرت محاسباتی بالاتر :** یعنی مسائل بزرگ را با دقت بالا در زمان کم بتوان حل کرد.

۱.۳.چرا پردازش موازی ؟

به نظر می رسد تلاش برای افزایش عملکرد کامپیوتر دیجیتال بی پایان باشد. در دو دهه گذشته عملکرد ریزپردازنده ها از رشد نمایی برخوردار بوده است.در هر ۱۸ ماه ریز پردازنده ها با فاکتور ۲ به لحاظ عامل (سرعت/عملکرد) رشد داشته اند.(و یا درحدود ۶۰% در سال) که این مسئله به عنوان قانون مور مشهور است . این رشد ناشی از ترکیبی از دو عامل زیر است :

۱- افزایش در پیچیدگی تراشه های VLSI (ناشی از تراکم بالاتر و اندازه بزرگتر دستگاه) ، پیش بینی افزایش به حدود ۱۰ میلیون ترانزیستور در یک تراشه برای هرریزپردازنده ، و ۱ بیلیون خانه حافظه با امکان دستیابی تصادفی (^{DRAM}) در سال ۲۰۰۰.
۲- معرفی، و بهبود در، ویژگی های معماری از قبیل حافظه های Cache ، بافرهای بزرگ ، دستوالعملهای چندگانه در چرخه ، چند نخی (multi threading) ، خطوط لوله عمیق (deep pipelines) ، اجرای خارج از ترتیب دستورالعملها ، و پرش به مکانی نامعلوم .

قانون مور در اصل در سال ۱۹۶۵ با مضمون دو برابرشدن پیشرفت و پیچیدگی تراشه درهر سال (که بعدا به هر ۱۸ ماه تجدید نظر شده) بر اساس تعداد کمی داده تدوین شد. پیش بینی مور تقریبا کاملا واقعی افزایش درتعداد ترانزیستورها در تراشه های ریزپردازنده و رم را نشان می دهد.
به نظر می رسد قانون مور صرفنظر عملکرد پردازنده مسایل زیر را مشخص میکند:
IPS:شمارش تعداددستورالعملهای اجرایی درهرثانیه (instructions per second)
FLOPS:شمارش تعدادعملیات ممیزشناوردرثانیه(floating-point per second)

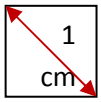


نمودار فوق پیش بینی رشد توان و کارایی پردازندها را در سنوات مختلف بر اساس قانون مور نشان می دهد.

محدودیتهای قانون مور :

- ✓ **محدودیت سرعت نور :** سرعت نور 3×10^8 می باشد که در فیبرنوری استفاده می شود ولی در تراشه ها از سرعت نور استفاده نمی شود . در تراشه ها از (انتشار سیگنال) استفاده می شود که یک سوم سرعت نور است و این نهایت سرعت است (1×10^8)

یک چیپ



$$t = \frac{v}{d} = \frac{\text{مسافت}}{\text{سرعت انتشار سیگنال}} = \frac{1\,cm}{1 * 10^8} = 0.1\,ns$$

یعنی در 0.1 ns یک دستورالعمل اجرا می شود حال در یک ثانیه 10×10^9 دستورالعمل اجرا می شود . که این نهایت سرعت با توجه به سرعت نور است .(این محدودیت را می توان با کم کردن مسافت چیپ کم کرد)

بنابراین باید با تکنیکهای موازی سازی سرعت را افزایش داد.(سخت افزار یک محدودیتی دارد)

در پردازش موازی به منظورافزایش سرعت از چندین پردازنده بصورت موازی برای حل مسائل استفاده می گردد.

چرا ما نیاز داریم که سرعت را افزایش دهیم ؟ (تا حد TIPS یا TFLOPS)

زیرا زمان اجرایی قابل قبول برابر است با کسری از چند ساعت (10^3-10^4 ثانیه)

در این زمان یک ماشین دارای سرعت **TIPS یا TFLOPS** می تواند 10^{15} تا 10^{16} عملیات را انجام دهد .

ثانیه	دستورالعمل
1	10^{12}
10^4	$x=10^{16}$

مثال : انتقال حرارت از اقیانوسهای جنوبی به قطب جنوب .

- ✓ اقیانوس به 4096 ناحیه از شرق به غرب 1024 ناحیه از شمال به جنوب و 12 لایه در عمق (50 میلیون سلول 3 بعدی)
- ✓ یک تکرار مدل شبیه سازی چرخه 10 دقیقه طول می کشد و 30 بیلیون عملیات نقطه اعشاری نیاز دارد.
- ✓ برای شبیه سازی در یک سال تقریبا 50 هزار تکرار نیاز است
- ✓ شبیه سازی برای 6 سال 10^{16} عملیات اعشاری نیاز دارد.

مثال : محاسبات سیال دینامیک

- ✓ حجم مورد مطالعه یک مشبک با اندازه $10^3 \times 10^3 \times 10^3$ نقطه مدل سازی می شود.
- ✓ درهرمرحله نیاز به 10^3 عملیات اعشاری روی نقطه در زمان 10^4 نیاز دارد.
- ✓ تعداد کل عملیات ها به 10^6 عملیات اعشاری می رسد.

تعریف دقیق پردازش موازی : انجام چندین عملیات در زمان واحدرا موازی سازی گویند.

مثال : پیدا نمودن اعداد اول بین ۱ تا N با استفاده از الگوریتم غربال نمودن اعداد.(برای مثال اعداد ۱ تا ۳۰)

- ✓ ابتدا لیست اعداد 1,2,3,...,N را داریم و می خواهیم اعداد اول آنرا جدا کنیم .
- ✓ بدین منظور از یک بردار بیت N خانه ای (بردار بیت MARK)استفاده می کنیم (مقدار اولیه این بردار صفر می باشد)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- ✓ درهر مرحله ، عدد بعدی بدون مارک m (با بیت صفر در موقعیت m ام از مارک)
- ✓ پیدا نمودن عنصر m و مارک نمودن (یک نمودن بیت متناظر mark) با شروع m^2

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

- ✓ سپس m بعدی را که بیت مارک آن صفر است را پیدا می کنیم که در این مثال عدد ۳ است. حال اعداد مضرب ۳ را که اول است غربال می کنیم .

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	1	1	1	0	1	0	1	1	1	0	1	0	1	1	1	0	1	0	1	1	1	0	1	0	1	1	1	0	1

- ✓ سپس m بعدی را که بیت مارک آن صفر است را پیدا می کنیم که در این مثال عدد ۵ است. حال اعداد مضرب ۵ را که اول است غربال می کنیم .

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	1	1	1	1	1	0	1	1	1	0	1	0	1	1	1	0	1	0	1	1	1	0	1	1	1	1	1	0	1

- ✓ سپس m بعدی را که بیت مارک آن صفر است را پیدا می کنیم که در این مثال عدد ۵ است. حال اعداد مضرب ۷ را که اول است غربال می کنیم .

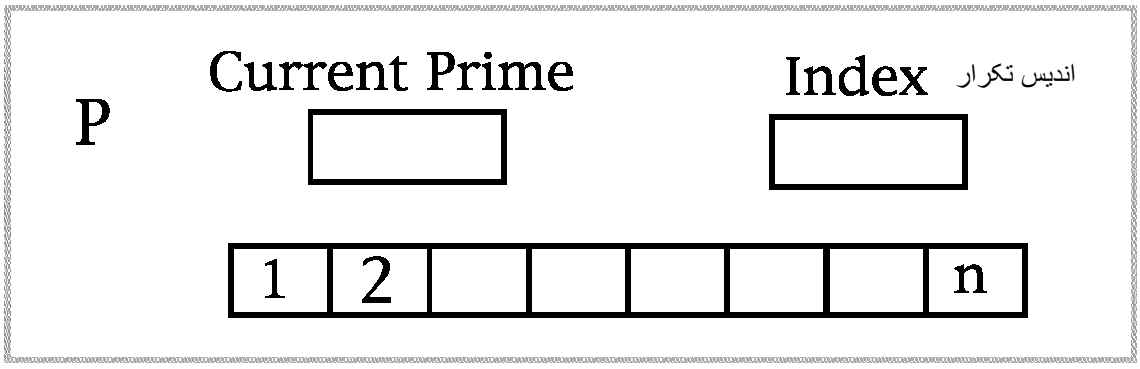
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	0	1	0	1	1	1	0	1	1	1	1	1	0	1

.
.
.

- ✓ الگوریتم تا موقعی ادامه می یابد که $m^2<n$ باشد

حال اگر این الگوریتم را در یک پردازنده انجام دهیم . خواهیم داشت :

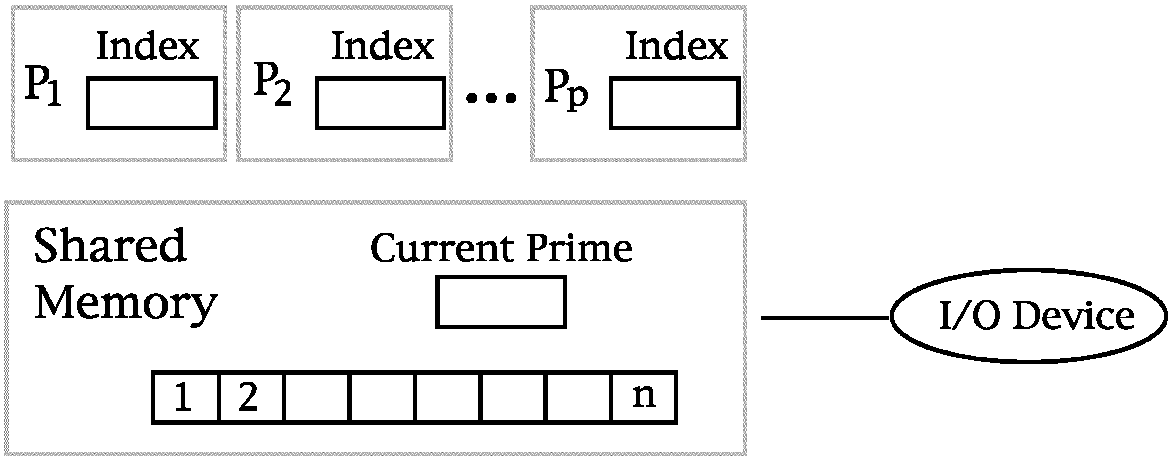
Single-Processor Implementation of the Sieve



مقدار اولیه=$M^2=(\text{Current Prime})^2$

اگر الگوریتم را بر روی P پردازنده انجام دهیم خواهیم داشت :

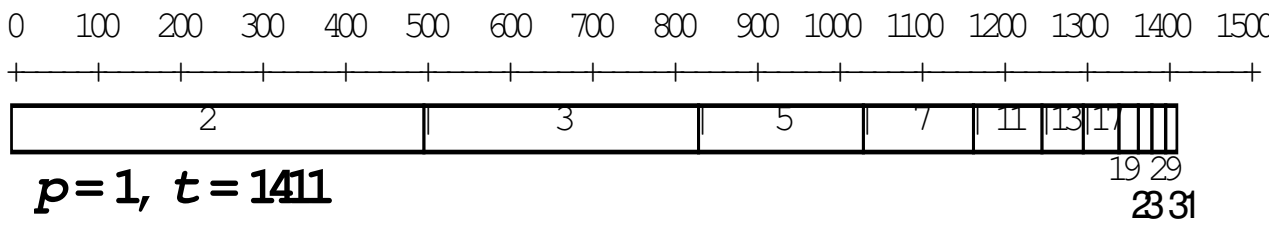
Control-Parallel Implementation of the Sieve



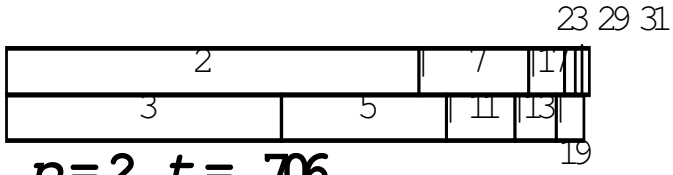
حافظه اشتراکی چند پردازنده ای شامل آرایه mark و Current prime=m است.

- ✓ P پردازنده به طور موازی با شروع $Index=m^2$ اعداد مضرب m که mark آنها صفر باشد را پیدا نموده و آرایه mark را تغییر می دهند.

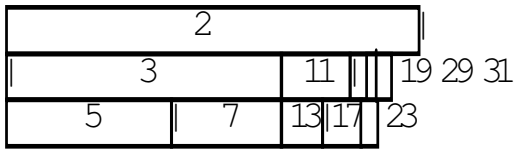
حال به مقایسه زمان سپری شده برای یافتن اعداد اول در مثال فوق در دو وضعیت پردازش سری (تک پردازنده ای) و موازی (چند پردازنده ای) می پردازیم :



$p = 1,\ t = 1411$



$p = 2,\ t = 706$

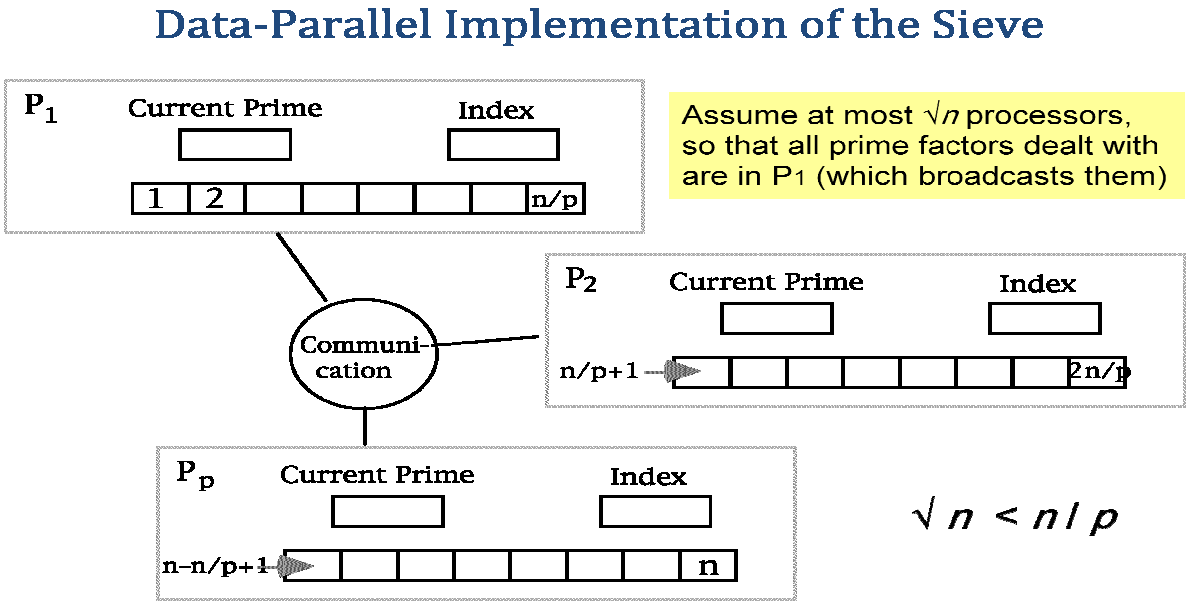


$p = 3,\ t = 499$

همانطور که در نمودار فوق ملاحظه می گردد در یک سیستم تک پردازنده ای (نمودار اول) ابتدا مضارب ۲ بررسی شده و سپس مضارب ۳ ، مضارب ۵ ، مضارب ۷ ، مضارب ۱۱ ، مضارب ۱۳ ، مضارب ۱۷ ، مضارب ۱۹ ، مضارب ۲۳ ، مضارب ۲۹ و در نهایت مضارب ۳۱ بررسی میگردند که این پروسه زمان ۱۴۱۱ را به خود اختصاص می دهد . ولی در سیستم ۲ پردازنده ای به موازات بررسی مضارب ۲ در پردازنده اول ، مضارب ۳ نیز می تواند در پردازنده دوم پردازش گردد و چنین است برای مضارب بعدی لذا مشاهده می گردد زمان کل پردازش در این سیستم کمتر(۷۰۶) از زمان تک پردازنده ای می باشد .

همچنین در نمودار سوم که مشخص کننده عملیات روی یک سیستم ۳ پردازنده ای است زمان کمتری(۴۹۹) صرف خواهد شد.

تکنینک دیگر برای موازی سازی مثال فوق تقسیم بردار بیتی Mark به P قطعه (به اندازه تعداد پردازنده ها) معادل هر قطعه در حافظه پردازنده ذخیره می شود .



۱.۴ طبقه بندی انواع موازی سازی

- دسته بندی M.J.Flynn در سال ۱۹۵۵

۱. SISD : تک دستوالعملی تک پردازنده ای (یک دستورالعمل روی یک دیتا انجام شود)
۲. SIMD : چند پردازنده ای که توسط دستورات صادرشده توسط یک واحد کنترل هدایت می شود.(یک دستوالعمل روی چند دیتا انجام شود)
۳. MISD : چند دستورالعمل روی یک دیتا انجام شود .
۴. MIMD : یک تعداد پردازنده داریم که روی یک تعداد دیتا پردازش را انجام می دهند. که خود دارای انواع زیر می باشد :
۴.۱. GMSV : (Global Memory_Shared Variables)
۴.۲. GMMP : (Global Memory_Massage Passing)
۴.۳. DMSV : (Distributed Memory_ Shared Variables)
۴.۴. DMMP : (Distributed Memory_ Massage Passing)

۱.۵. موانع پردازش موازی

- ✓ **قانون گروش (Grosch) :** هزینه تحقیقاتیP پردازنده معادل هزینه P^2 در یک سیستم تک پردازنده ای است
- ✓ **قانون مینسکی (Minsky) :** افزایش سرعت متناسب با $\log_2 p$ می باشد(p تعداد پردازنده ها می باشد)
- ✓ **فناوری های IC :** در هر ۵ سال سرعت ۱۰ برابر میشود.
- ✓ **قانون امدل (Amdahl's) :** کد(دستورات) غیرموازی (ترتیبی) سرعت را به شدت محدود می کند.

$$s = \frac{1}{f+(1-f)/p} \leq \min(p, 1/f)$$

F : درصد غیرموازی کد (ترتیبی)

1-f : درصد موازی کد

موازی سازی فقط بستگی به تعداد پردازنده ها ندارد بلکه به کدی که قرار است موازی شود نیز دارد.

۱.۶. ارزیابی سیستمهای پردازش موازی

به منظور ارزیابی پارامترهای زیر را تعریف می کنیم :

P : تعداد پردازنده ها

W(p) : تعداد کل واحدهای عملیاتی انجام گرفته توسط P پردازنده یا کار محاسباتی با انرژی یک پردازنده

T(p) : زمان اجرایی با P پردازنده

$T(p) <= W(p)$ $W(1) = T(1)$

S(p) : افزایش سرعت (قانون امدل) $T(1)/T(p)$

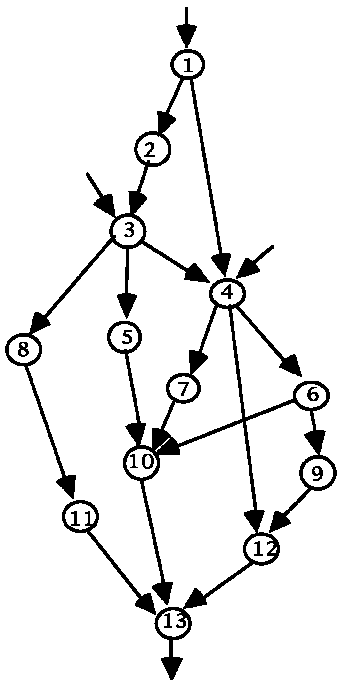
E(p) : بهره وری $T(1) / [P \times T(p)]$

R(p) : سخت افزاری که بصورت اضافی استفاده شده است $W(p) / W(1)$

Q(p) : کیفیت $T^3(1) / [P \times T^2(P) \times W(p)]$

مثال : گراف وظیفه داده شده زیر را در نظر بگیرید . اگر زمان اجرای هر وظیفه یک واحد باشد پارامترهای W(1) و T(1) و T(∞) را بدست آورید .

1.6 Effectiveness of Parallel Processing

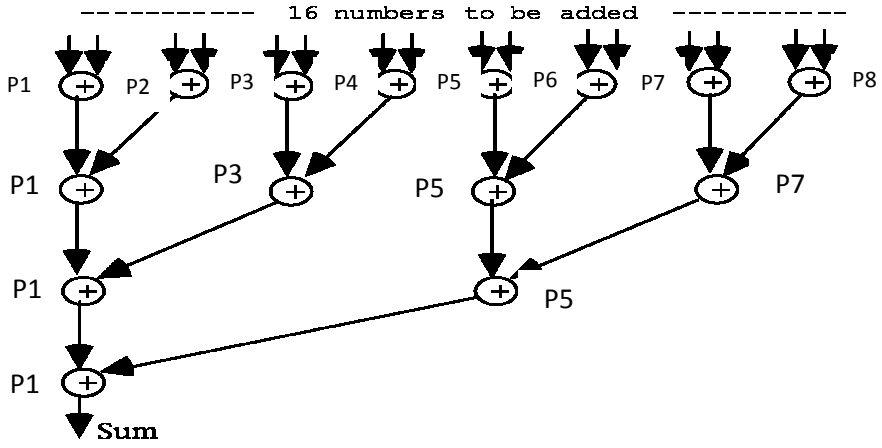


W(1)=13 : تعداد کل واحدهای عملیاتی انجام گرفته توسط ۱ پردازنده برابر ۱۳ می باشد

T(1)=13 : زمان اجرایی با ۱ پردازنده برابر است با ۱۳ (W(1)=T(1))

T(∞)=8 : چون گراف فوق دارای ۸ سطح می باشد سرعت در موازی سازی بیشتر از ۸ نمی شود.

مثال : جمع ۱۶ عدد در یک سیستم چند پردازنده ای با ۸ پردازنده (P=8)



الف) اگرهزینه ارتباطی بین پردازنده ها را صفر در نظر بگیریم پارامترهای T(1)وW(1) وT(p) وW(p) وS(p) و E(p) و R(p) و Q(p) را بدست آورید.

W(1)=15 : تعداد کل واحدهای عملیاتی انجام گرفته توسط ۱ پردازنده .

T(1)=15 : زمان اجرایی با ۱ پردازنده . ((W(1)=T(1))

W(8)= هزینه ارتباطی + تعداد کل محاسبات جمع = 15 + 0 = 15

T(8)= تعداد سطوح گراف = 4

S(8)=T(1)/T(8) : افزایش سرعت ⟵ S(8)= 15/4=3.75

E(8)=T(1)/[8 x T(8)] : بهره وری ⟵ E(8)= 15/[8x4]=%47 (یعنی 52% ازپردازنده ها بیکارند)

R(8)=W(P)/W(1) : ⟵ R(8)= 15/15=1

Q(8)= T^3(1) / [8 x T^2(8) x W(8)] : کیفیت ⟵ Q(8)= (15)^3/[8x(4)^{2}x15]=1.76

ب) اگر هزینه ارتباطی بین پردازنده ها را یک درنظر بگیریم پارامترهای T(1) و W(1) و T(p) و W(p) و S(p) و E(p) و R(p) و Q(p) را بدست آورید.

W(1)=15 : تعداد کل واحدهای عملیاتی انجام گرفته توسط ۱ پردازنده .

T(1)=15 : زمان اجرایی با ۱ پردازنده . ((W(1)=T(1))

W(8)= هزینه ارتباطی + تعداد کل محاسبات جمع = 15 + 7 = 22

T(8)= تعداد سطوح گراف + طولانی ترین مسیر = 4+3=7

S(8)=T(1)/T(8) : افزایش سرعت ⟵ S(8)= 15/7=2.14

E(8)=T(1)/[8 x T(8)] : بهره وری ⟵ E(8)= 15/[8x7]=%27 (یعنی 73% ازپردازنده ها بیکارند)

R(8)=W(P)/W(1) : ⟵ R(8)= 22/15=1.47

Q(8)= T^3(1) / [8 x T^2(8) x W(8)] : کیفیت ⟵ Q(8)= (15)^3/[8x(7)^{2}x22]=0.39

فصل دوم از قسمت اول

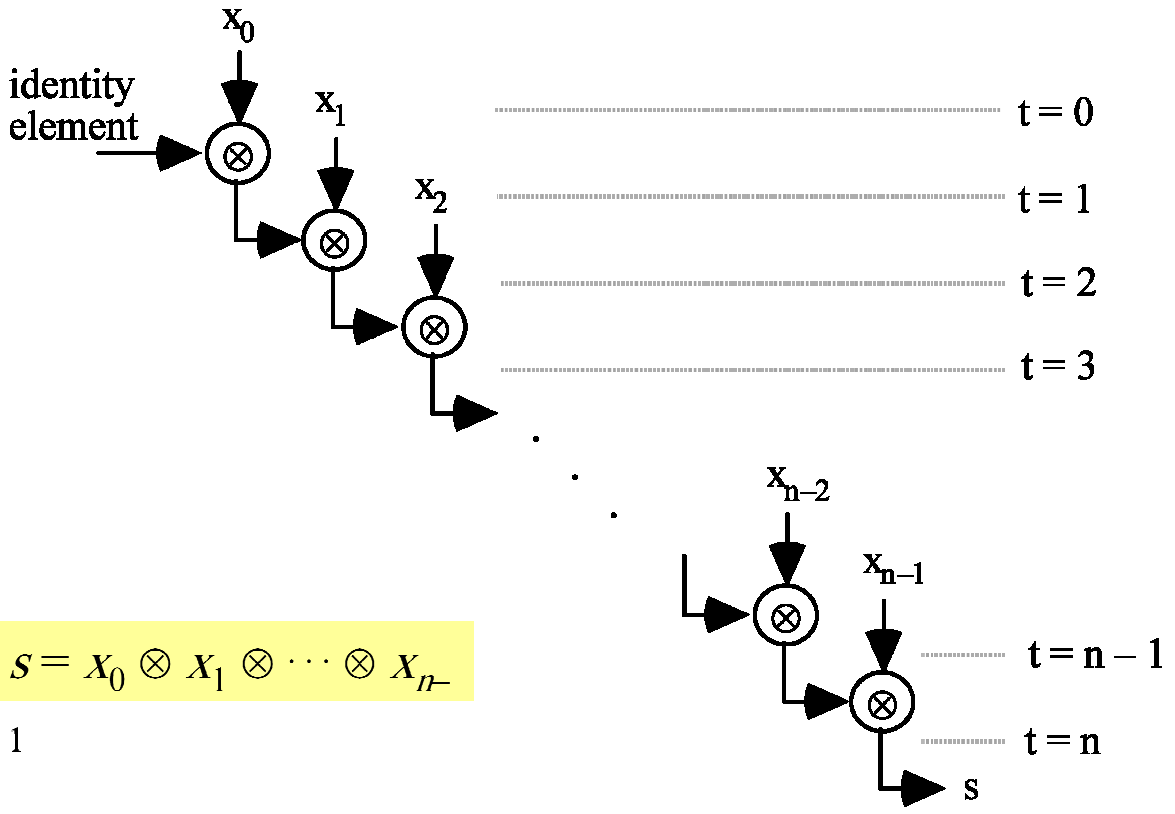
نگاهی سریع به برخی از الگوریتم های موازی:

در این قسمت اطلاعاتی در مورد ماهیت الگوریتم های موازی و پیچیدگی آنها بدست می آوریم با بررسی ۵ نوع از محاسبات موازی با ۴ نوع از معماریهاری موازی ساده(۲۰ نوع ترکیب) :

- ✓ که ۵ نوع محاسبات موازی عبارتند از :
  - محاسبات زنجیره ای (Semigroup)
  - محاسبات موازی (Prefix)
  - مسیریابی بسته ای (PacKet Routing)
  - انتشاری (Broad casting)
  - مرتب سازی رکوردها (Sorting)

۲.۱. برخی محاسبات ساده :

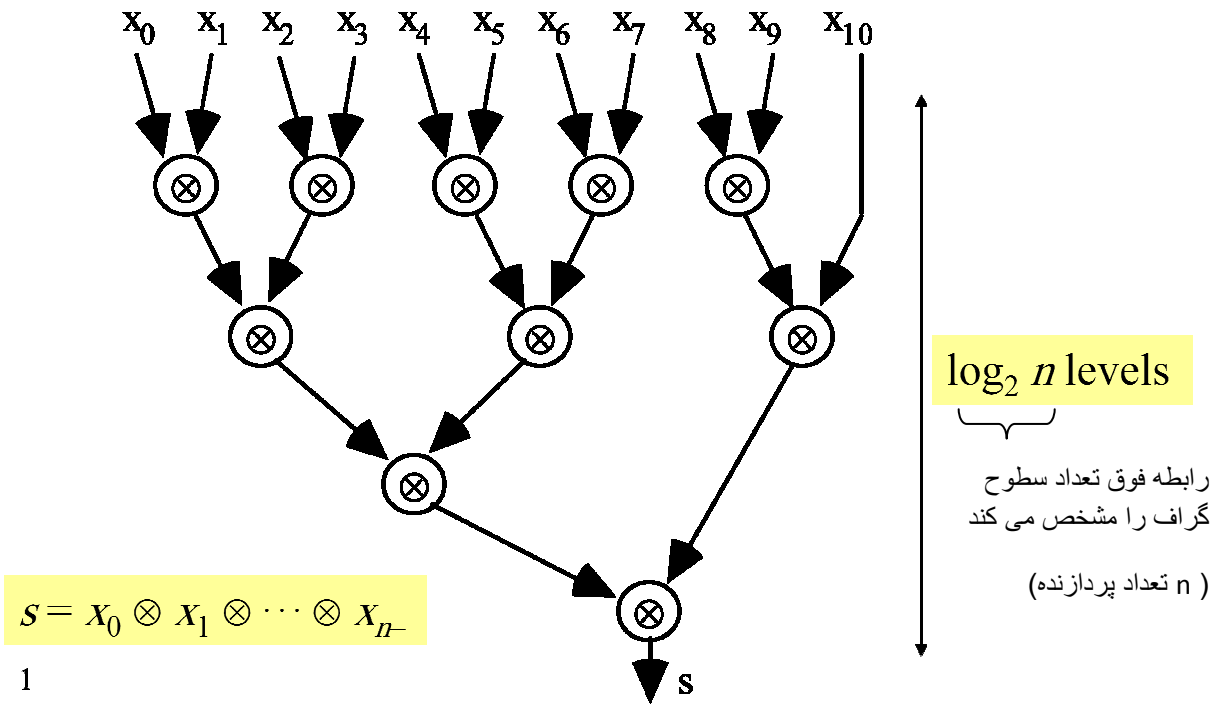
شکل زیر پیاده سازی Semigroup (محاسبات زنجیره ای) را در یک سیستم تک پردازنده ای نشان می دهد .



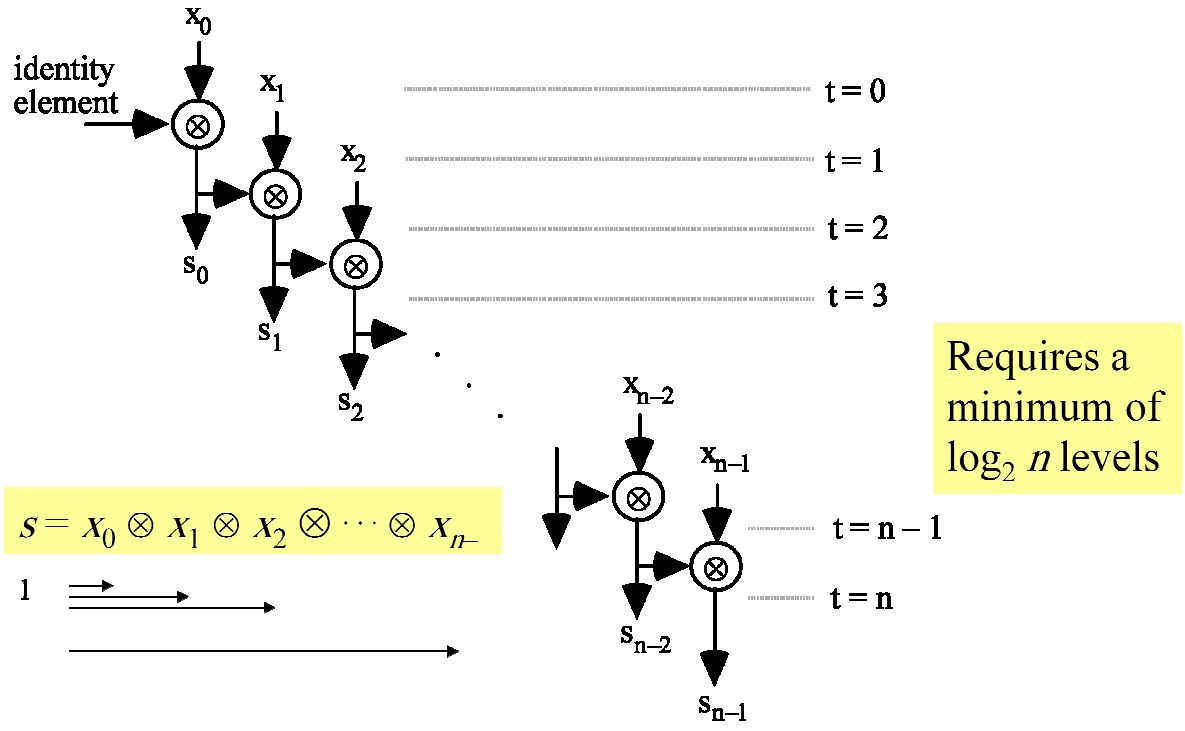
$$s = x_0 \otimes x_1 \otimes \cdots \otimes x_{n-1}$$

$\otimes$: این عملگر میتواند یکی از عملیات (+ یا x یا AND یا OR یا XOR یا ∪ یا ∩ یا Max یا Min) باشد.

شکل زیر پیاده سازی Semigroup (محاسبات زنجیره ای) را در یک سیستم چند پردازنده ای و بصورت موازی نشان می دهد .



شکل زیر پیاده سازی محاسبات موازی Prefix را در یک سیستم تک پردازنده ای نشان می دهد .



محاسبات موازی Prefix همانند Semigroup است با این تفاوت که نتایج میانی هم مورد نیاز است (S_0 و S_1 و ...)

۲.۲. برخی معماری های ساده :

پارامترهای زیر در بررسی معماریها مطرح می باشند .

- ✓ قطر پردازنده (D) : تفاوت بین طولانی ترین و کوتاهترین مسیر بین یک زوج پردازنده .
- ✓ حداکثر درجه گره (d) : حداکثر تعداد پیوندها یا کانالهای ارتباطی برای یک پردازنده .

۲.۲.۱. پردازنده هایی که بصورت آرایه ای خطی بهم متصل می باشند :

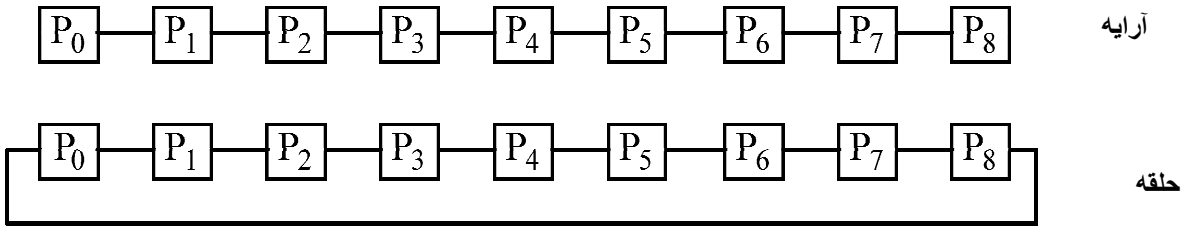


Fig. 2.2 A linear array of nine processors and its ring variant.

در مدل آرایه **D=P-1** بوده و در مدل حلقه **$D=\lfloor p/2 \rfloor$** می باشد.

لذا برای شکل فوق که P=9 می باشد در مدل آرایه D=9-1=8 بوده و برای مدل حلقه **$D=\lfloor 9/2 \rfloor=4$**

۲.۲.۲. پردازنده هایی که به صورت درخت دودویی به هم متصلند :

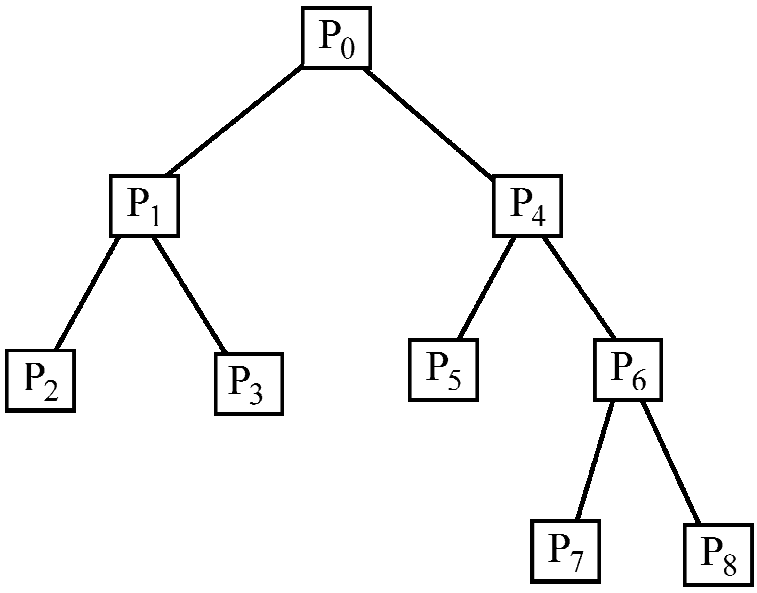
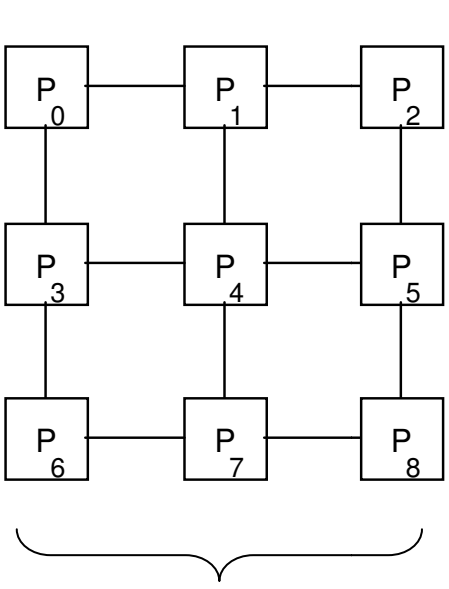


Fig. 2.3 A balanced (but incomplete) binary tree of nine processors.

d=3

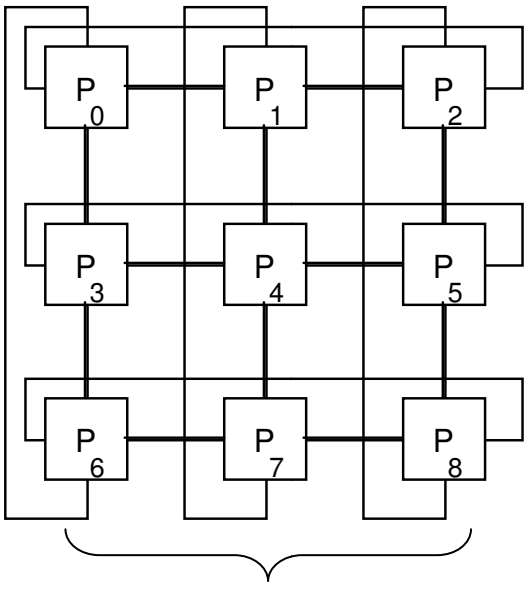
$D=2\lfloor \log_2 p \rfloor=2\lfloor \log_2 9 \rfloor=6-1=5$

۲.۲.۳. پردازنده هایی که به صورت مش دو بعدی (2D Mesh) به هم متصلند :



A

$d = 4$
$D = 2\sqrt{p} - 2$
$B \cong \sqrt{p}$



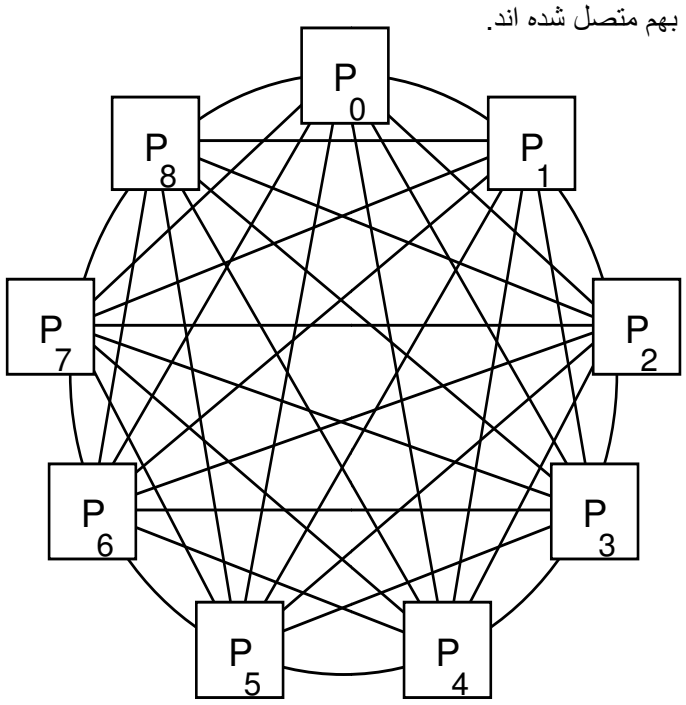
B

$(\sqrt{p})$
$(2\sqrt{p})$

❖ مزیت معماری نوع B به معماری نوع A اینست که مقدار پارامتر D را کاهش داده است .

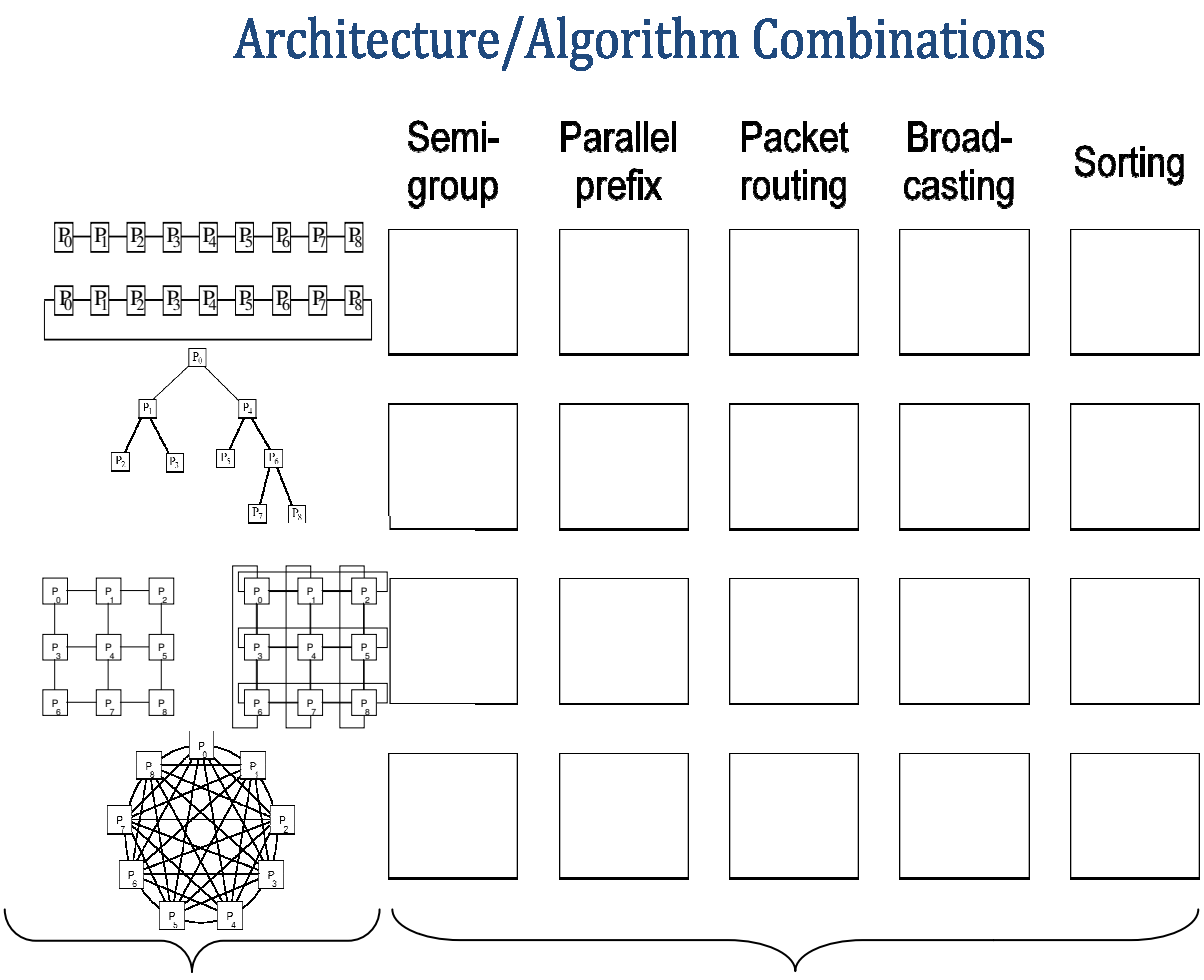
۲.۲.۴. پردازنده هایی که به صورت معماری حافظه اشتراکی (Shared-Memory) به هم متصلند :

در این نوع از معماری همه پردازنده ها ۲ به ۲ بهم متصل شده اند.



$d = p - 1$
$D = 1$
$B = \lfloor p/2 \rfloor \lceil p/2 \rceil$

پس از معرفی ۴ نوع معماری (خطی – درخت باینری – مش دوبعدی – حافظه اشتراکی) حال به ترکیب این معماریها با ۵ الگوریتم (Semigroup – Parallel Prefix – Packet routing – Broadcasting - Sorting) خواهیم پرداخت :



۲.۳. الگوریتم هایی برای معماری خطی linear array :

۲.۳.۱. ترکیب اول : الگوریتم Semigroup برای یک معماری خطی linear array :

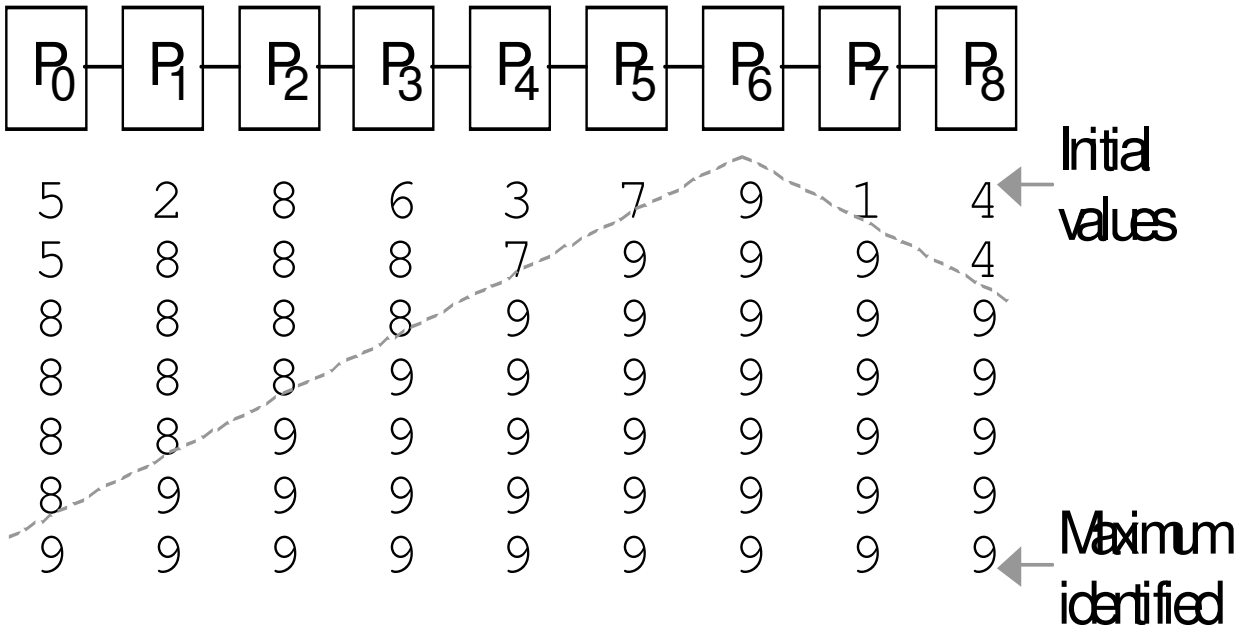
- ✓ در الگوریتم Semigroup نسخه آخر مهم است (نتیجه) محاسبات میانی مهم نیستند.

مثال : فرض کنید یک معماری آرایه خطی با ۹ پردازنده داریم (P_0-P_8) ودر هر پردازنده یک عدد ذخیره شده است . هدف پیدا کردن ماکزیمم مقدار این اعداد است .

- ✓ هر پردازنده در هر مرحله مقدار ماکزیمم خود را به دو همسایه مجاور خود ارسال می کند .
- ✓ هر پردازنده با دریافت مقادیر پردازنده ها ی مجاور ، ماکزیمم را به صورت رابطه زیر بدست می آورد.

Max(Left , Own , Right)

- ✓ این الگوریتم زمانی خاتمه می یابد که هر پردازنده از همسایه های خود ۲ عدد یکسان دریافت کند. (همه پردازنده ها عدد یکسان دریافت کنند)

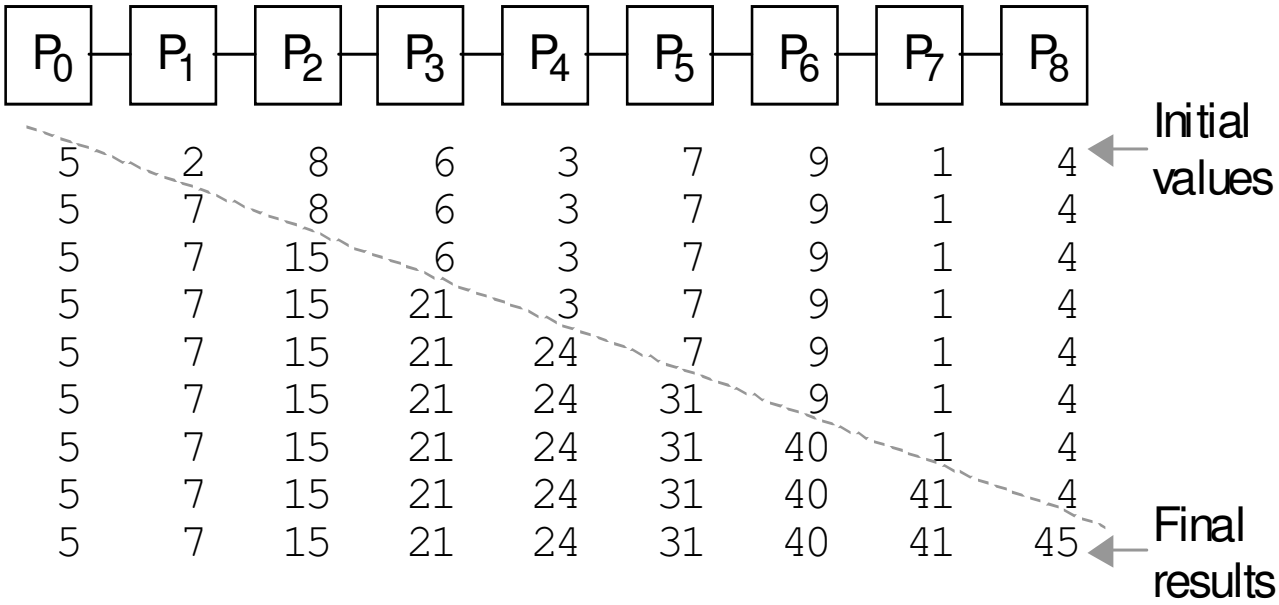


۲.۳.۲ ترکیب دوم : الگوریتم Prefix برای یک معماری خطی linear array :

- ✓ در الگوریتم Prefix جوابهای میانی نیز مورد نیاز است.

مثال : فرض کنید یک معماری آرایه خطی با ۹ پردازنده داریم (P_0-P_8) ودر هر پردازنده **یک عدد** ذخیره شده است . هدف محاسبه حاصل جمع این اعداد است .

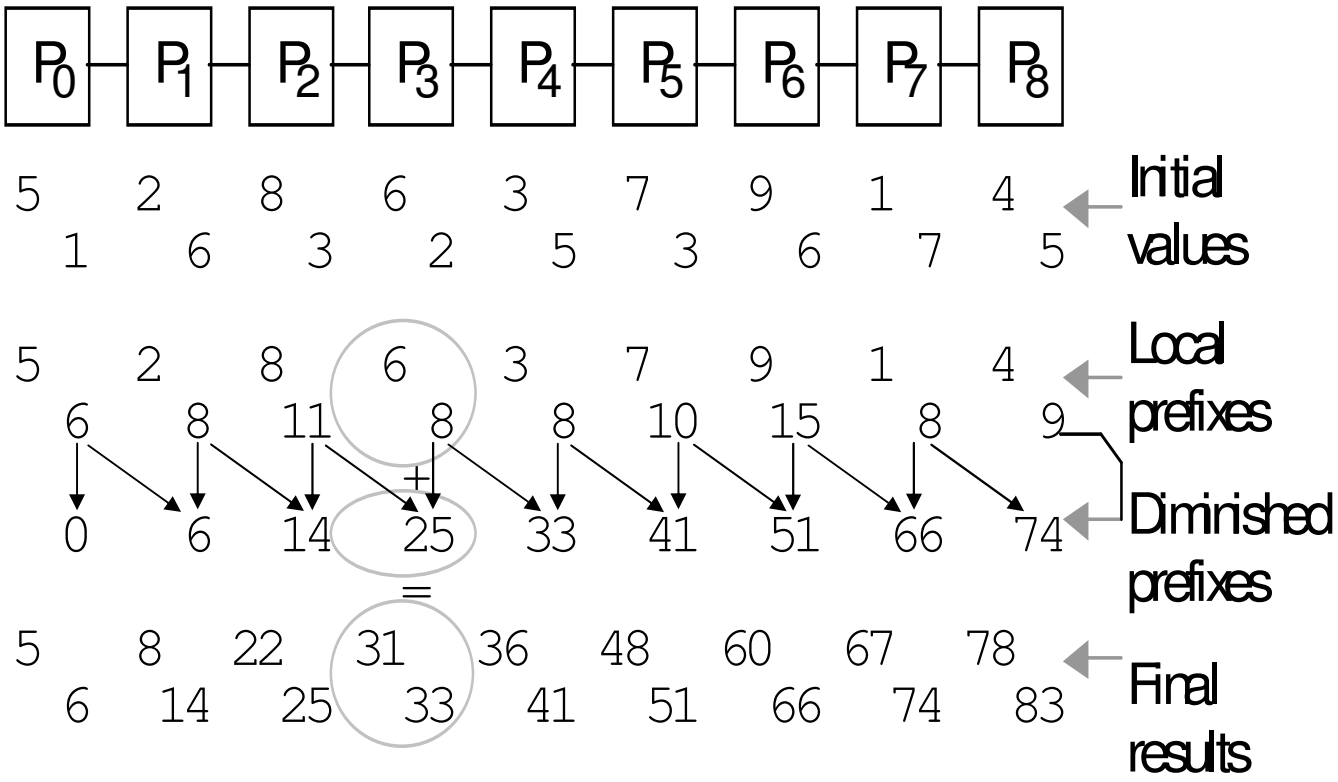
- ✓ در هر مرحله یک پردازنده مقدار خود را با مقدار پردازنده سمت چپ خود جمع کرده و در خود ذخیره می کند .
- ✓ در هر مرحله فقط یک پردازنده عمل جمع را انجام می دهد و بقیه پردازنده ها منتظر می مانند تا جمع انجام شده به پایان برسد .
- ✓ عمل جمع از اولین پردازنده آغاز و در مرحله بعدی تاثیر خود را برروی پردازنده بعدی (به عنوان ورودی جمع) می گذارد.
- ✓ این الگوریتم در آخرین پردازنده به پایان می رسد.



❖ **نکته :** درمثال فوق تفاوتی به لحاظ زمان اجرا بین یک سیستم ۹ پردازنده ای و ۱ پردازنده ای وجود ندارد.

مثال : فرض کنید یک معماری آرایه خطی با ۹ پردازنده داریم (P_0-P_8) ودر هر پردازنده **دو عدد** ذخیره شده است . هدف محاسبه حاصل جمع این اعداد است .

- ✓ محاسبه مجموعه داده ها در هر پردازنده
- ✓ انجام عملیات جمع بصورت Prefix
- ✓ انجام عملیات جمع نتایج مراحل فوق



شرح حل مثال فوق :

Initial Values :همانطورکه مشاهده میگرددهرپردازنده دارای ۲ مقدارمی باشدمثلا P_0 دارای دومقدار 5 و 1 است .

یا P_1 دارای دومقدار 2 و 6 می باشد و ...

Local Prefixes : در این مرحله بصورت همزمان هر پردازنده ۲ مقدار خود را جمع می کند. مثلا جمع مقادیر پردازنده P_0 که 6=1+5 و یا جمع مقادیر پردازنده P_1 که 8=6+2 و ... که همگی همزمان انجام می گردد.

Diminished Prefixes : در این مرحله حاصل جمع های محاسبه شده در مرحله قبل (حاصل جمعهای درونی هر پردازنده) بصورت سری و غیر همزمان (همانند مثال قبل) با هم جمع می گردد.

Final result : در این مرحله حاصل جمع های محاسبه شده در مراحل قبل (**Local Prefixes** و **Diminished Prefixes**) با هم جمع می گردد.

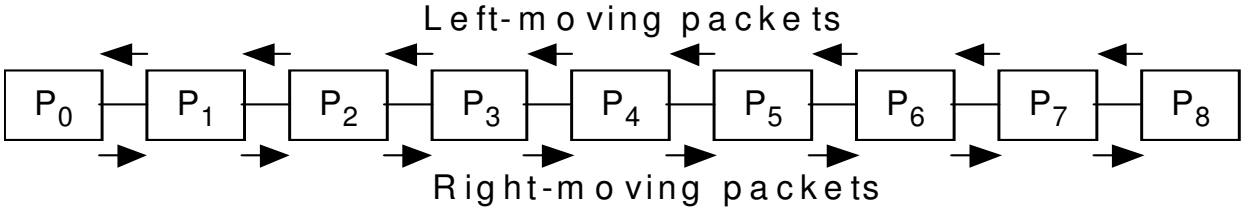
۲.۳.۳ ترکیب سوم : الگوریتم مسیریابی بسته ای (Packet routing) برای یک معماری خطی linear array :

پردازنده P_i میخواهدبسته ای را به P_j ارسال کند.محاسبه (j-i)برای تعیین فاصله وجهت که به آن Routing tag گویند. اگر این مقدار مثبت (+) باشد حرکت به سمت راست (Right) بوده و اگر این مقدار منفی (-) باشد حرکت به سمت چپ (Left) می باشد

مثال : P_1 می خواهد به P_7 بسته ای ارسال کند : Routing tag = 7 – 1 = +6 پس حرکت به سمت راست است.

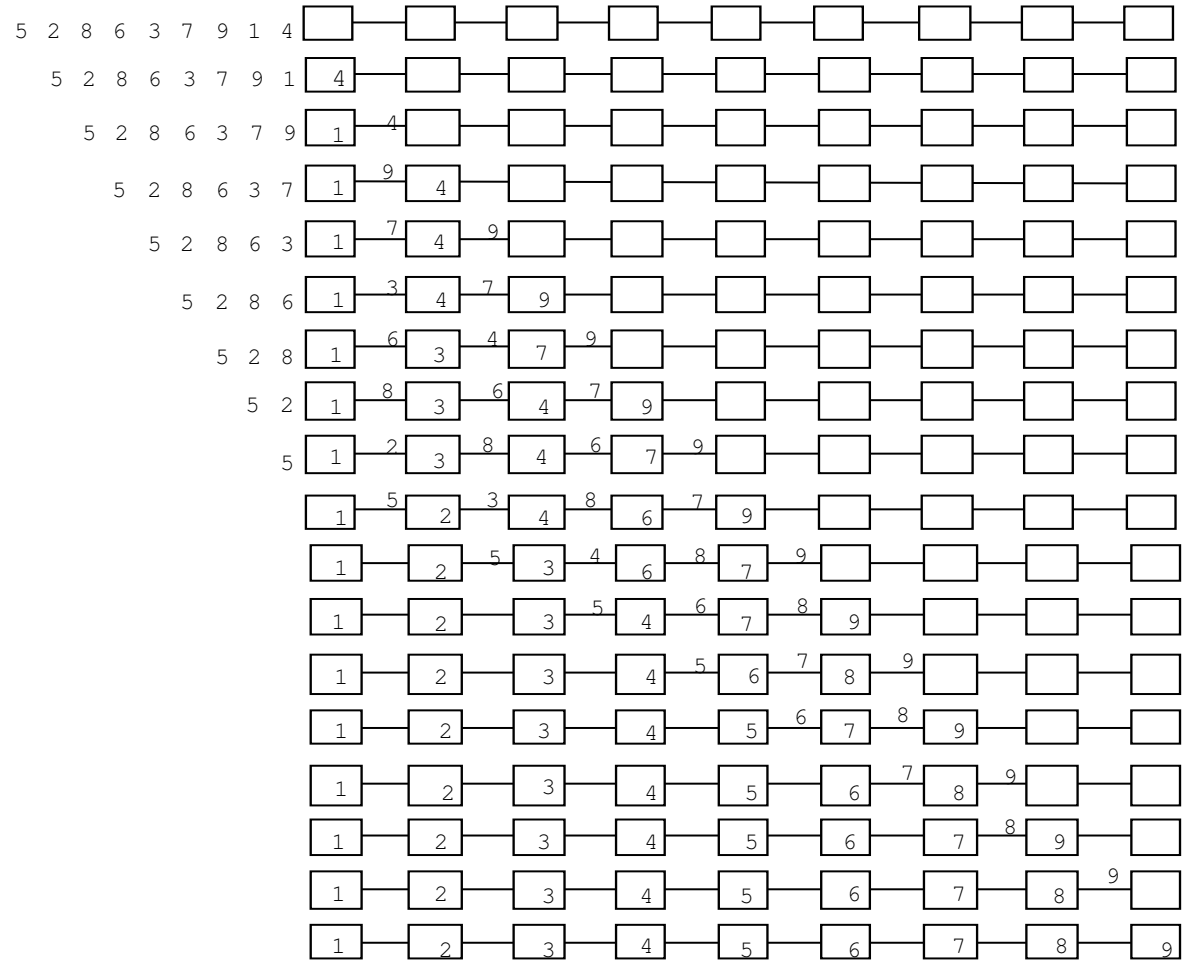
۲.۳.۴ ترکیب چهارم : الگوریتم مسیریابی انتشار (Broadcasting) برای یک معماری خطی linear array :

در حالت انتشاری اگر پردازنده P_i بخواهد بسته ای را به تمامی پردازنده ها ارسال کند ، آن پیام را به صورت rbcast(a) به همسایه سمت راست و پیام Lbcast(b) را به همسایه سمت چپ خود ارسال می کند.



۲.۳.۵ ترکیب پنجم : الگوریتم مرتب سازی (Sorting) برای یک معماری خطی linear array :

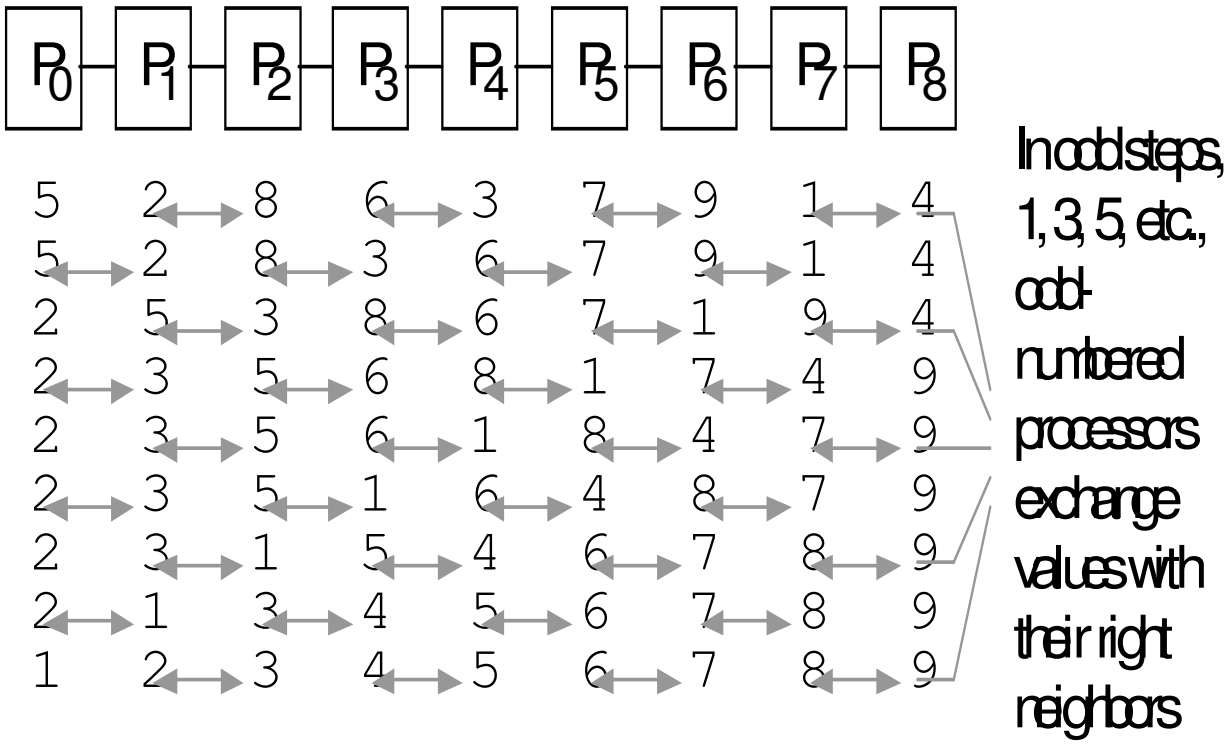
- ✓ **مرتب سازی بر اساس کلید خارجی (Externally Supplied Keys)**
- مقدار اولیه پردازنده ها ∞ است .
- هرپردازنده یک مقدار کلید از سمت چپ دریافت می کند و آنرا با مقدار ذخیره شده در رجیستر داخل خود مقایسه می کند. کوچکترین مقدار را نگه می دارد و مقدار بزرگتر را به پردازنده سمت راست خد انتقال می دهد.



✓ **مرتب سازی بر اساس کلید داخلی (Internally Stored Keys)**

این نوع از مرتب سازی را اصطلاحا (Odd-even transposition) نیز گویند که دارای ۲ مرحله Odd و Even می باشد.

- **مرحله Odd-number :** در این مرحله پردازنده های فرد مقادیر خود را با همسایه های زوج سمت راست مقایسه می کنند ، اگر مقادیر فوق مرتب نباشند داده های خود را جابه جا می کنند.
- **مرحله Even-number :** در این مرحله پردازنده های زوج مقادیر خود را با همسایه های فرد سمت راست مقایسه می کنند ، اگر مقادیر فوق مرتب نباشند داده های خود را جابه جا می کنند.



تشریح مثال فوق :

مرحله Odd-number : ابتدا بطور همزمان P_1 مقدار خود را با P_2 و P_3 مقدار خود را با P_4 و P_5 مقدار خود را با P_6 و P_7 مقدار خود را با P_8 مقایسه میکند و اگر لازم باشد مقادیر جابه جا می گردند.

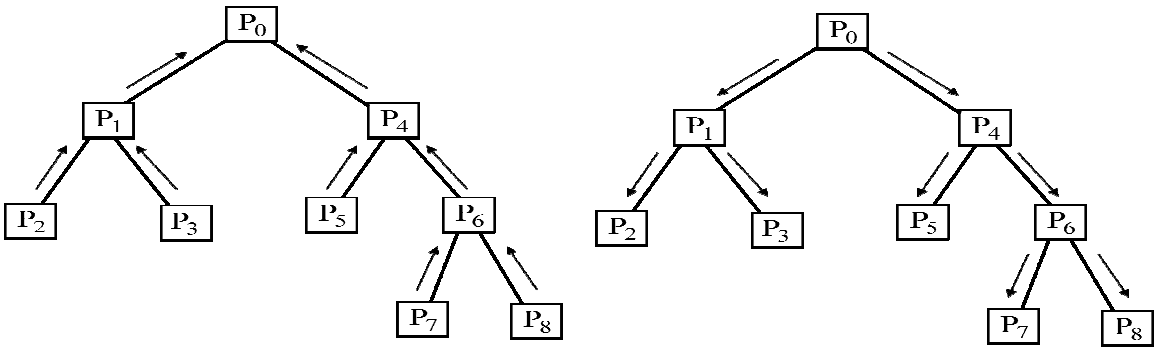
مرحله Even-number : سپس بطورهمزمان P_0 مقدار خود را با P_1 و P_2 مقدار خود را با P_3 و P_4 مقدار خود را با P_5 و P_6 مقدار خود را با P_7 مقایسه میکند و اگر لازم باشد مقادیر جابه جا می گردند.

عملیات فوق تا زمانی ادامه میابد که هیچ جابه جایی صورت نپذیرد.

۲.۴. الگوریتم هایی برای درخت دودیی Binary Tree :

۲.۴.۱. ترکیب اول : الگوریتم Semigroup برای یک معماری دودیی Binary Tree :

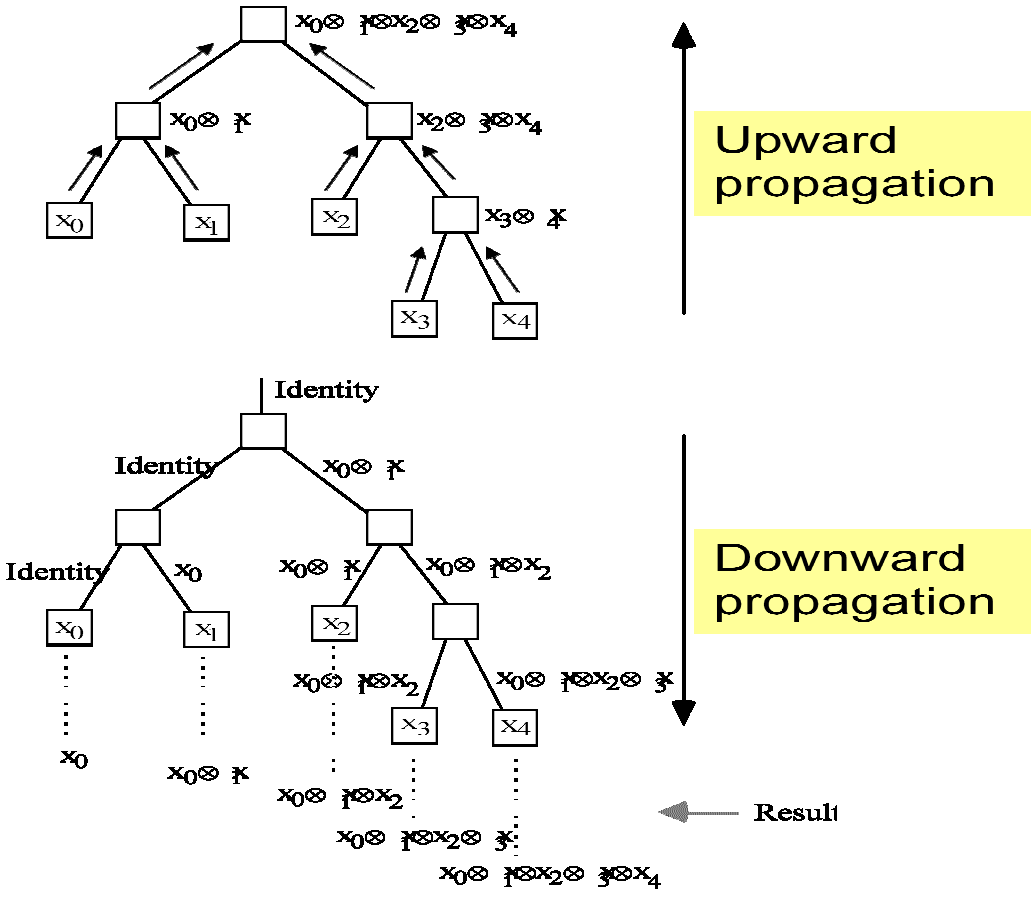
- ✓ برگهای درخت داده های X_0 تا X_{n-1} قرار می گیرد.
- ✓ انتشار از پایین به بالا



۲.۴.۲. ترکیب دوم : الگوریتم محاسبات موازی Prefix برای یک معماری دودیی Binary Tree :

- ✓ **فاز ۱ :** انتشار از پایین به بالا (Semigroup)
- ✓ **فاز ۲ :** انتشار از بالا به پایین

در فاز ۱ هر گره داده ای را که از فرزند سمت چپ خود دریافت می کند ذخیره می کند.



مثالهایی برای نشان دادن سودمندی محاسبت Prefix :

مثال ۱ : لیستی از صفرها و یکها داریم و می خواهیم ردیف (Rank) یکها را مشخص کنیم .

Ranks of 1s in a list of 0s/1s:

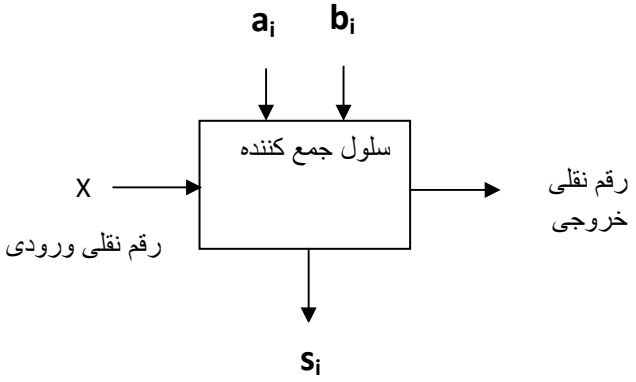
Data:	0	0	1	0	1	0	0	1	1	1	0
Prefix sums:	0	0	1	1	2	2	2	3	4	5	5
Ranks of 1s:			1		2			3	4	5	

مثال ۲ : یک مدار اولویت از یک لیست صفرها و یکها تشکیل شده است و هدف انتخاب پایین ترین اولویت اولین '1' در لیست است .

Priority arbitration circuit:

Data:	0	0	1	0	1	0	0	1	1	1	0
Dim'd prefix ORs:	0	0	0	1	1	1	1	1	1	1	1
Complement:	1	1	1	0	0	0	0	0	0	0	0
AND with data:	0	0	1	0	0	0	0	0	0	0	0

مثال ۳ : محاسبه رقمهای نقلی در یک جمع کننده با محاسبات موازی Prefix :



رقم نقلی خروجی ۳ حالت دارد :

g : اگر نتیجه جمع a_i+b_i بزرگتر از ۹ باشد رقم نقلی تولید می گردد .

p : اگر نتیجه جمع a_i+b_i مساوی ۹ باشد رقم نقلی انتشارمی یابد .

a : اگر نتیجه جمع a_i+b_i کمتر از ۹ باشد رقم نقلی از بین می رود .

Carry-lookahead network:

$p ¢ x = x$ X روی P انتشار می یابد

$a ¢ x = a$ X از بین می رود

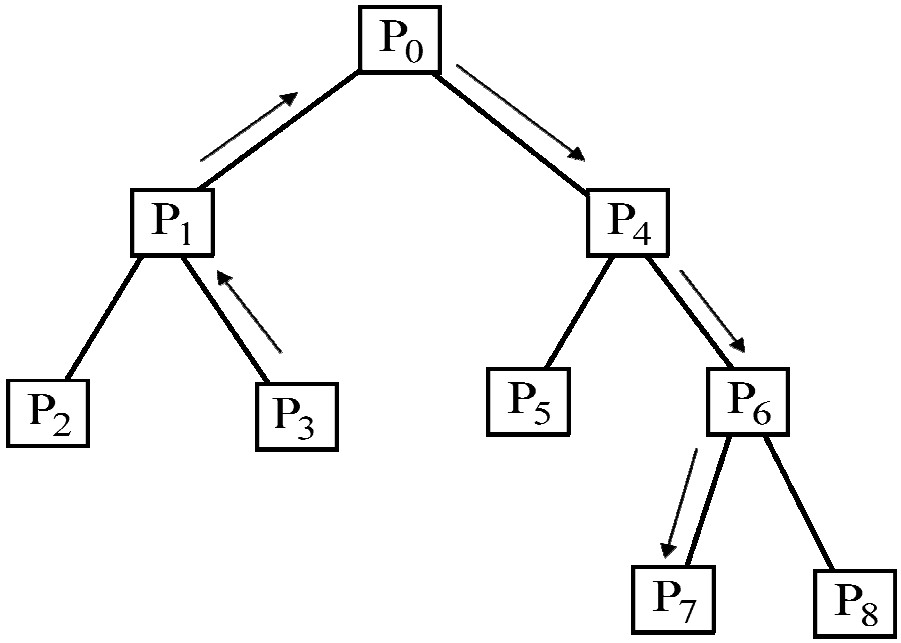
$g ¢ x = g$ g تولید می شود

¢ : عملگر رقم نقلی

۲.۴.۳. ترکیب سوم : الگوریتم مسیریابی بسته Packet Routing برای یک معماری دودیی Binary Tree :

هدف ارسال یک بسته از پردازنده **i** ام به پردازنده **j** ام است .

- ✓ الگوریتم مسیریابی بستگی به طرح شماره گذاری پردازنده ها در درخت باینری دارد .
- ✓ فرض می کنیم که طرح شماره گذاری "Preorder" باشد.



الگوریتم :

خودش باشد مقصد

```
1-if dest = self
2-then remove the packet {dor
3-else if dest < self or dest > ma
4-      then route upward
5-      else if dest δ maxl
6-           then route leftward
7-           else route rightward
8-           endif
9-      endif
10-endif
```

maxl : بزرگترین شماره گره در سمت چپ (در گره ای که قرار داریم)

maxr **:** بزرگترین شماره گره در سمت راست (درگره ای که قراردارایم)

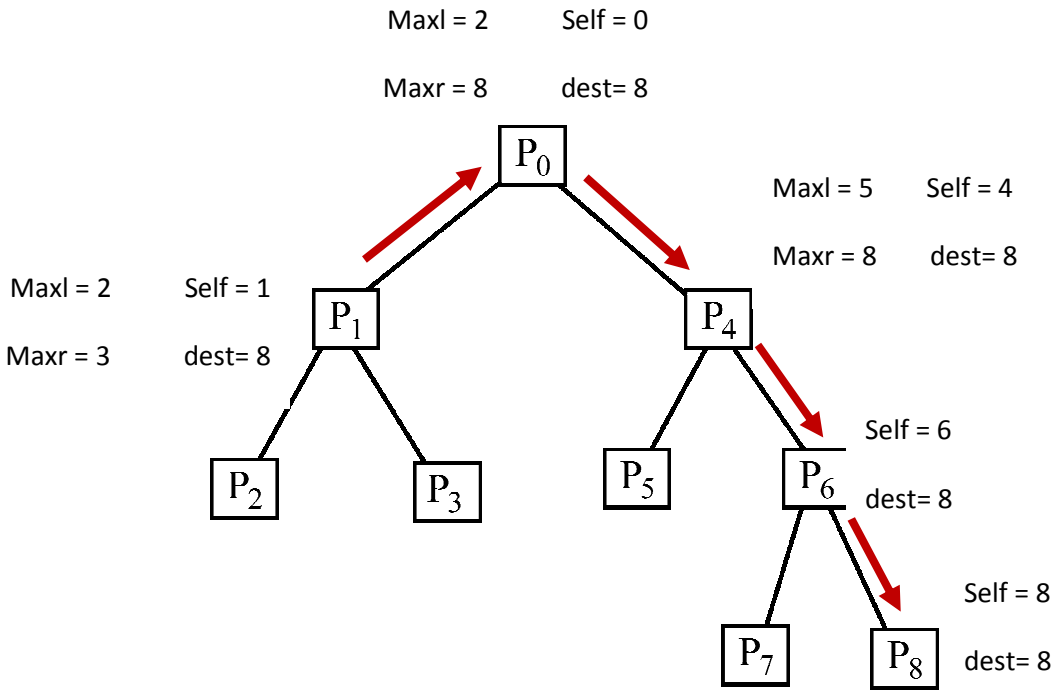
شرح الگوریتم :

اگر گره مقصد همان گره مبدا است الگوریتم پایان یافته و بسته حذف می گردد

در غیر اینصورت اگر شماره گره مقصد کوچکتر از گره مبدا بوده یا شماره گره مقصد بزرگتر از بزرگترین شماره گره در سمت راست باشد به سمت بالا حرکت (مسیر یابی) می کند

در غیر اینصورت اگر شماره گره مقصد کوچکتریا مساوی بزرگترین شماره گره در سمت چپ باشد به سمت چپ حرکت (مسیر یابی) می کند درغیر اینصورت به سمت راست حرکت (مسیر یابی) می کند.

مثال : ارسال بسته ای از پردازنده شماره ۱ به پردازنده شماره ۸ :



دراین مثال نقطه شروع حرکت بسته P_1 و مقصد P_8 می باشد لذا در نقطه P_1 متغیر Self=1 (شماره گره) و dest=8 (مقصد) که در تمامی مسیر دارای مقدار شماره گره مقصد بوده و Maxl=2 (چون در گره ۱ بزرگترین شماره گره در شاخه چپ گره شماره ۲ می باشد) و Maxr=3 (چون در گره ۱ بزرگترین شماره گره در شاخه راست گره شماره ۳ می باشد) . لذا بر اساس خط شماره ۳ الگوریتم حرکت (مسیریابی) به سمت بالا (گره شماره 0) انجام می شود.

حال در گره P_0 هستیم و متغیرها به صورت (Maxl=2 , Maxr=8 , Self=0 , dest=8) تغییر میکنند. بنابراین طبق خط شماره ۷ الگوریتم (if dest > maxl) به سمت راست (گره شماره ۴) حرکت می کنیم .

حال در گره P_4 هستیم و متغیرها به صورت (Maxl=5 , Maxr=8 , Self=4 , dest=8) تغییر میکنند. بنابراین طبق خط شماره ۷ الگوریتم (if dest > maxl) به سمت راست (گره شماره ۶) حرکت می کنیم .

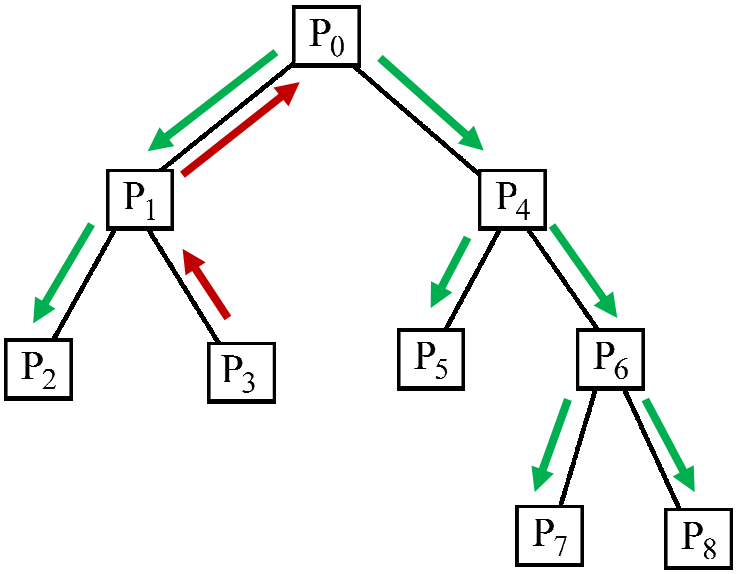
حال در گره P_6 هستیم و متغیرها به صورت (Maxl=7 , Maxr=8 , Self=6 , dest=8) تغییر میکنند. بنابراین طبق خط شماره ۷ الگوریتم (if dest > maxl) به سمت راست (گره شماره ۸) حرکت می کنیم .

حال در گره P_8 هستیم و متغیرها به صورت (Maxl=8 , Maxr=8 , Self=8 , dest=8) تغییر میکنند. بنابراین طبق خط شماره ۱ الگوریتم (if dest = self) الگوریتم پایان پذیرفته و بسته به مقصد رسیده است .

۲.۴.۴. ترکیب چهارم : الگوریتم انشاری یسته Brodcasting برای یک معماری دودیی Binary Tree :

پردازنده **i** ام داده مورد نظر را به پردازنده ریشه به پایین به بالا (Upward) ارسال می کند و بعد از آن بسته به تمامی پردازنده ها بصورت بالا به پایین (downward) ارسال می گردد.

مثال : ارسال بسته ای از پردازنده شماره ۳ به کلیه پردازنده ها (انتشار)



۲.۴.۵. ترکیب پنجم : الگوریتم مرتب سازی Sort برای یک معماری دودیی Binary Tree :

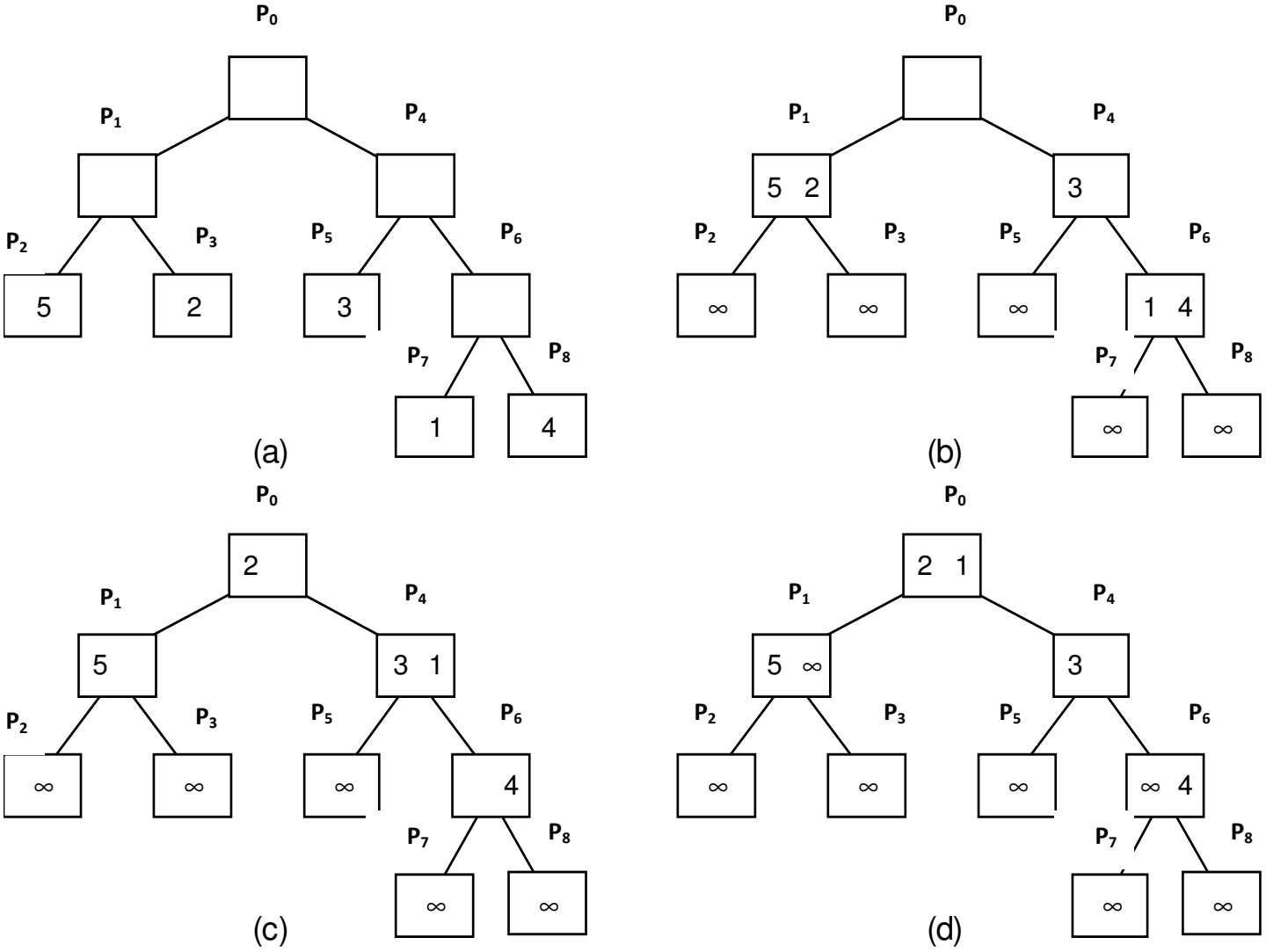
- ✓ در ابتدا هر برگ درخت یک داده دارد و گره های دیگر خالی است.
- ✓ هر گره داخلی فضای ذخیره سازی برای دو داده را دارد .
- ✓ عملیات Upward (پایین به بالا) از زیر درختهای چپ به راست انجام می گیرد.

شبه کد نوشته شده برای الگوریتم مرتب سازی : (این الگوریتم مشابه سورت حبابی است)

1-if you have 2 items *اگر هر گره ۲ آیتم داشته باشد آنگاه عملیاتی انجام نمی گیرد (پایان)*
2-then do nothing
3-else if you have 1 item that came from the left (right)
4- then get the smaller item from the right (left) child
5- else get the smaller item from each child
6- endif
7-Endif

در غیر اینصورت اگرگره دارای یک ایتم باشد که از چپ به راست آمده است آنگاه عنصر کوچکتر را

مثال : مطلوب است پیاده ساز ی الگوریتم سورت روی درخت باینری زیر :



(a) : وضعیت ابتدایی درخت را نشان می دهد که اعداد نامرتب برروی برگها قرار داشته و دیگر گره ها دارای مقدار ∞ می باشند.

(b) : پردازنده P_1 ۲ عنصر را در خود جای می دهد که در ابتدای کار مقادیر آن ∞ می باشد. در این مرحله ۵ و ∞ با هم مقایسه شده و چون ۵ از ∞ کوچکتر است جابه جا شده و همین عملیات برای P_1 و P_3 اتفاق می افتد که چون ۲ از ∞ کوچکتر است جابه جایی صورت می پذیرد بنابرای مقدار P_1 از (∞و∞) به (5,2) تغییر می یابد. و همین عملیات بین P6 و P7 و سپس بین P6 و P8 و در نهایت بین P4 و P5 انجام میگردد تا عناصر کوچکتر به بالای درخت انتقال یابند.

(c) : در این مرحله مقایسه بین P0 (با مقادیر ∞) و P1 (5,2) صورت می پذیرد که عنصر کوچکتر به بالا (P0) منتقل می گردد سپس مقایسه بین مقدار سمت راست P4 (∞) و مقادیر P6 (1,4) انجام شده که عنصر کوچکتر به بالا منتقل میگردد که این عنصر (۱) است.

(d) در مرحله آخر عنصر سمت راست P0 (∞)با مقادیر P4 (3,1) مقایسه شده و عدد کوچکتر به بالا منتقل می شود.

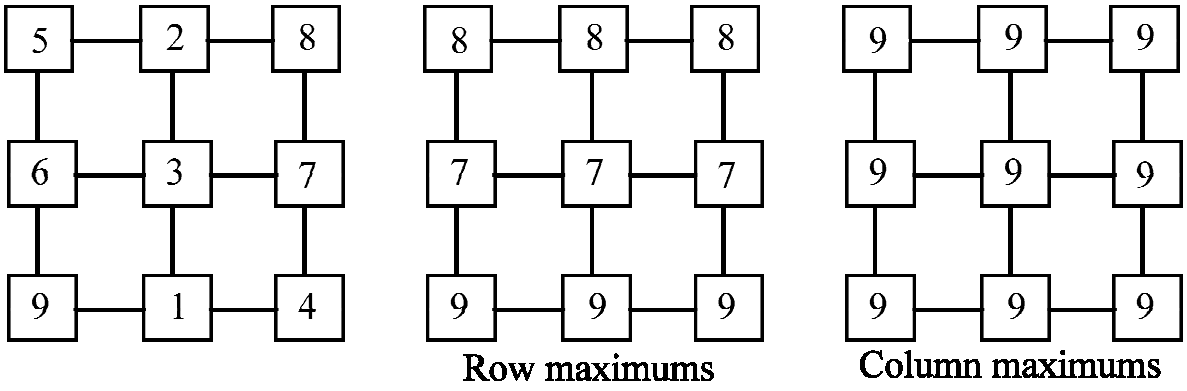
۲.۵. الگوریتم هایی برای معماری مش دو بعدی 2D Mesh :

۲.۵.۱. ترکیب اول : الگوریتم Semigroup برای یک معماری مش دو بعدی 2D Mesh :

- ✓ برای انجام محاسبات Semigroup محاسبات در هر سطر و سپس در هر ستون انجام می گیرد .

مثال : برای مثال محاسبه ماکزیمم یک مجموعه اعداد با P مقدار ، که در هر پردازنده یک مقدار ذخیره شده است .

- ✓ در ابتدا ماکزیمم سطرهای مش محاسبه شده و سپس از ستون ماکزیمم گیری می شود.



Finding the max value on a 2D mesh.

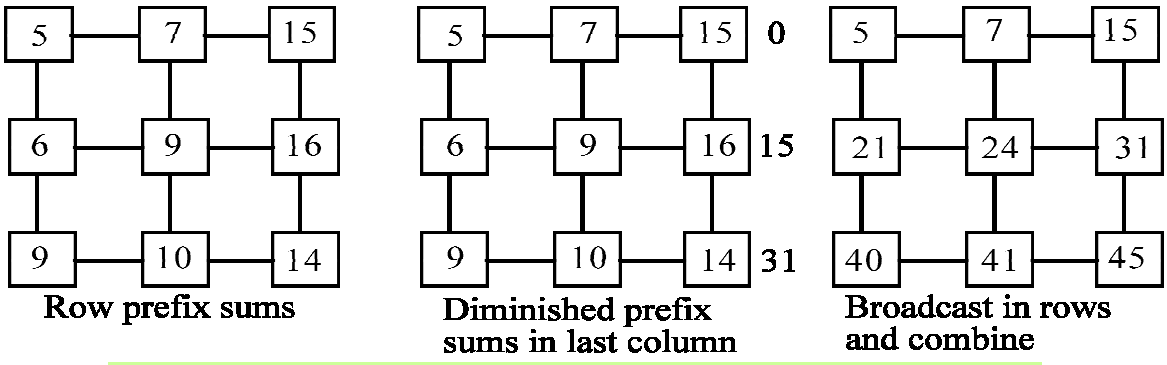
تعداد مراحل $4\sqrt{p} - 4$ **اگر P=9 باشد تعداد مراحل** $4\sqrt{9} - 4 = 12 - 4 = 8$

۲.۵.۲. ترکیب دوم : الگوریتم Parallel Prefix برای یک معماری مش دو بعدی 2D Mesh :

محاسبات در ۳ فاز انجام می گیرد (فرض کنید پردازنده ها به صورت مرتبه ردیفی (Row-major) اندیس گذاری می شوند)

- ✓ انجام محاسبت Prefix روی هر سطر به صورت مجزا
- ✓ انجام محاسبات Diminished Parallel Prefix در ستون سمت راست .
- ✓ انتشار نتایج ستون سمت راست به تمامی عناصر در ردیف های بعدی و ترکیب با مقادیر اولیه محاسبه شده مقادیر Row Prefix

مثال : برای مثال محاسبه حاصل جمع یک مجموعه اعداد با P مقدار ، که در هر پردازنده یک مقدار ذخیره شده است .

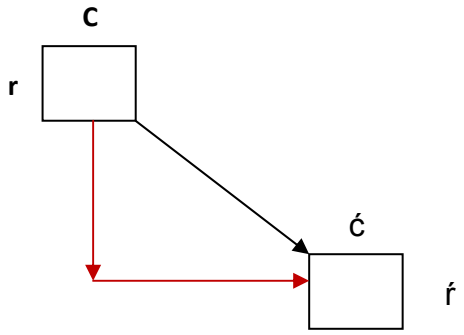


Computing prefix sums on a 2D mesh

۲.۵.۳. ترکیب سوم : الگوریتم مسیریابی Routing برای یک معماری مش دو بعدی 2D Mesh :

هدف ارسال بسته ای از پردازنده در موقعیت (r,c) به پردازده ای در موقعیت (ŕ,ć) است .

الگوریتم (Row-first-routing) : در این روش ابتدا بسته از ردیف r به ردیف ŕ ارسال و سپس بسته از ستون C به ستون ć ارسال می گردد.



۲.۵.۴. ترکیب چهارم : الگوریتم انتشار Broadcast برای یک معماری مش دو بعدی 2D Mesh :

انتشار در دو فاز انجام می گیرد :

- ✓ انتشار بسته به تمامی پردازنده ها در سطری که گره مبدا در آن قرار دارد .
- ✓ انتشار بسته به تمامی ستونها .

۲.۵.۵. ترکیب پنجم : الگوریتم مرتب سازی Shaer Sort برای یک معماری مش دو بعدی 2D Mesh :

مرتب سازی بجز برای مرحله آخر بصورت زیر انجام می گیرد :

فاز یک :

- ✓ سطرهای زوج (0,2,4,...) از چپ به راست مرتب سازی می شوند .
- ✓ سطرهای فرد (1,3,5,...) از راست به چپ مرتب سازی می شوند .
- ✓ تمامی ستون ها به طور مستقل از بالا به پایین مرتب سازی می شوند.

فاز دو :

- ✓ مرتب سازی سطرها از سمت چپ به راست .

تعداد تکرار فازها : $\lceil \log_2 r \rceil + 1$

مثال : مرتب سازی داده های مش زیر :

شماره سطر

0	5	2	8
1	6	3	7
2	9	1	4

تعداد فازها = $\lceil \log_2 3 \rceil + 1$ =2+1=3

حل : فاز یک : ابتدا سطرهای زوج را از چپ به راست مرتب می کنیم

شماره سطر

0	2	5	8
1	6	3	7
2	1	4	9

سپس سطرهای فرد را از راست به چپ مرتب می کنیم

0	2	5	8
1	7	6	3
2	1	4	9

سپس تمامی ستونها بطور مستقل از بالا به پایین مرتب سازی می شوند :

0	1	4	3
1	2	5	8
2	7	6	9

فاز یک را یک بار دیگر تکرار می کنیم :

ابتدا سطرهای زوج را از چپ به راست مرتب می کنیم

0	1	3	4
1	2	5	8
2	6	7	9

سپس سطرهای فرد را از راست به چپ مرتب می کنیم

0	1	3	4
1	8	5	2
2	6	7	9

سپس تمامی ستونها بطور مستقل از بالا به پایین مرتب سازی می شوند :

0	1	3	2
1	6	5	4
2	8	7	9

فاز دو : مربب سازی سطرهای از چپ به راست :

0	1	2	3
1	4	5	6
2	7	8	9

۲.۶. الگوریتم هایی برای معماری حافظه اشتراکی Shared-Memory :

۲.۶.۱. ترکیب اول : الگوریتم Semigroup برای یک معماری حافظه اشتراکی Shared-Memory :

هر پردازنده می تواند به صورت محلی محاسبات خود را انجام دهد.

۲.۶.۲. ترکیب دوم : الگوریتم Parallel Prefix برای یک معماری حافظه اشتراکی Shared-Memory :

همانند Semigroup می باشد با این تفاوت که داده ها از پردازنده های با شماره کوچکتر ترکیب می شوند.

۲.۶.۳. ترکیب سوم : الگوریتم مسیریابی Routing برای یک معماری حافظه اشتراکی Shared-Memory :

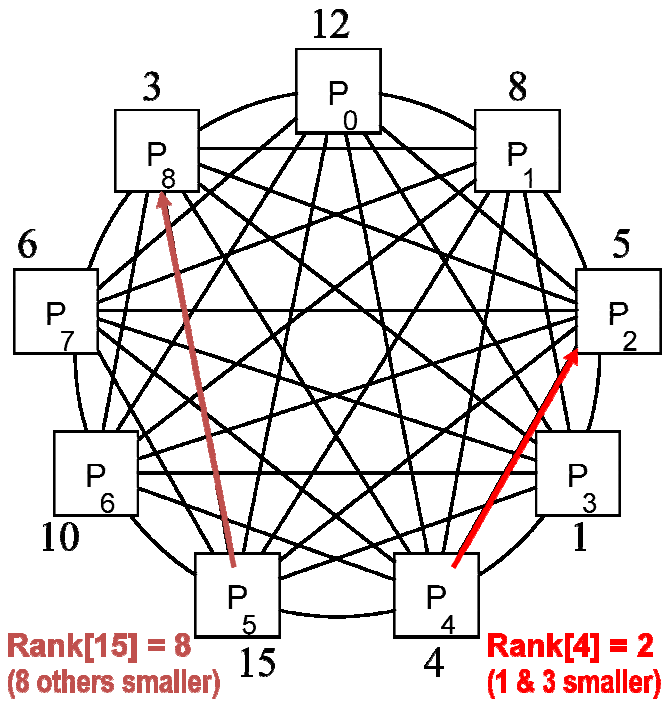
کم اهمیت است.

۲.۶.۴. ترکیب چهارم : الگوریتم اتشار Broadcast برای یک معماری حافظه اشتراکی Shared-Memory :

یک پردازنده با تمام پردازنده ها در ارتباط است .

۲.۶.۵. ترکیب پنجم:الگوریتم مرتب سازی Shaer Sort برای یک معماری حافظه اشتراکی Shared-Memory :

هر پردازنده با مسیریابی رتبه داده در اختیار خود را تعیین می کند.



Semigroup computation: Each processor can perform the computation locally

Parallel prefix computation: Same as semigroup, except only data from smaller-index processors are combined

Packet routing: Trivial

Broadcasting: One step with all-port ($p-1$ steps with single-port) communication

Sorting: Each processor determines the rank of its data element; followed by routing

فصل سوم از قسمت اول

پیچیدگی الگوریتم های موازی : (این قسمت توسط استاد بصورت خلاصه تدریس شد**)**

مروری بر پیچیدگی الگوریتم و کلاس های پیچیدگی های مختلف :

- ✓ معرفی مفاهیمی چون **"زمان"** و بهینه سازی **"زمان/هزینه"**
- ✓ معرفی ابزاری برای تجزیه و تحلیل ، مقایسه ، و تنظیمات ریز.

۳.۱. پیچیدگی تحلیل asymptotic (نزدیک به واقع) :

تحلیل الگوریتم : دراین بخش می خواهیم به تجزیه و تحلیل الگوریتمها به لحاظ زمان اجرای یک الگوریتم بپردازیم .

تحلیل الگوریتم به ۲ صورت **"دقیق"** و **"تقریبی"** انجام می گیرد.

- ✓ **تحلیل دقیق الگوریتم :**محاسبه تعداد دفعات انجام عملیات و حافظه مصرفی یک الگوریتم بصورت دقیق .
- ✓ **تحلیل تقریبی الگوریتم :** به این نوع تحلیل اصطلاحا **asymptotic (**نزدیک به واقع**)** گویند.

برای بیان پیچیدگی **asymptotic** فرض می کنیم که ۳ نوع الگوریتم مرتب سازی داده ها (Sort) با زمان مصرفی زیر وجود دارد (**n** اندازه مسئله)

1) $(\log_2 n)^2$

2) $(\log_2 n)^2/2 + 2\log_2 n$

3) $500\ \log_2 n$

برای بحث روی ۳ الگوریتم فوق(جهت تحلیل **asymptotic**) سه عملگر" O" ، "Ω" و "Θ" بصورت زیرتعریف می شوند :

:O $f(n) = O(g(n))\ \ if\ \ \exists\ C, n_0\ \ such\ \ That\ \forall n > n_0\ we\ have\ \ f(n) < C\ g(n)$

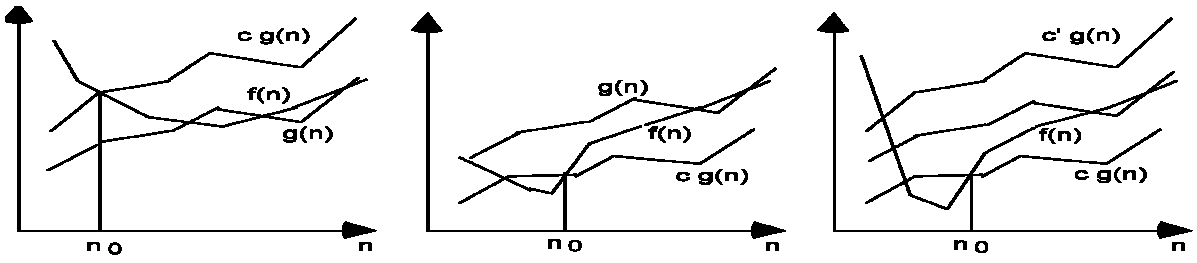
بدین مفهوم که f(n) سریعتر از g(n) است .

: "Ω" $f(n) = \Omega(g(n))\ \ if\ \ \exists\ C, n_0\ \ such\ \ That\ \forall n > n_0\ we\ have\ \ f(n) < C\ g(n)$

بدین مفهوم که رشد f(n) کمی سریعتر از g(n) است .

: "Θ" $f(n) = \theta(g(n))\ \ if\ \ \exists\ C, \acute{C}\ n_0\ \ such\ \ That\ \forall n > n_0\ we\ have\ \ C\ g(n) < f(n) < \acute{C}\ g(n)$

بدین مفهوم که f(n) و g(n) تقریبا با یک نرخ رشد پیدا می کنند.



$f(n) = O(g(n))$ $f(n) = \Omega(g(n))$ $f(n) = \Theta(g(n))$

$3n \log n = O(n^2)$ $\frac{1}{2}\, n \log^2 n = \Omega(n)$ $3n^2 + 200n = \Theta(n^2)$

تقسیم بندی سطوح کارایی یک الگوریتم :

- ✓ **Sub Linear :** هدف در پردازش موازی رسیدن به این سطح می باشد .
- ✓ **Linear (خطی) :** این سطح از الگوریتم مربوط به تک پردازنده ای می باشد ((O(n))
- ✓ **Super Linear**
- ❖ اگر با تکنیکها O(n) کم شود ، می شود Sub linear و اگر افزایش یابد می شود Super Linear .

شکل زیر نرخ رشد برخی از الگوریتم ها را که دارای ۳ وضعیت فوق می باشند نشان می دهد :

	Notation	Class name	Notes
Sub Linear	$O(1)$	Constant	Rarely practical
	$O(\log \log n)$	Double-logarithmic	Sublogarithmic
	$O(\log n)$	Logarithmic	
	$O(\log^k n)$	Polylogarithmic	k is a constant
	$O(n^a)$, $a < 1$		e.g., $O(n^{1/2})$ or $O(n^{1-\varepsilon})$
	$O(n/\log^k n)$		Still sublinear
	$O(n)$	Linear	
Super Linear	$O(n \log^k n)$		Superlinear
	$O(n^c)$, $c > 1$	Polynomial	e.g., $O(n^{1+\varepsilon})$ or $O(n^{3/2})$
	$O(2^n)$	Exponential	Generally intractable
	$O(2^{2^n})$	Double-exponential	Hopeless!

فصل چهارم از قسمت اول (مدلهای پردازش موازی)

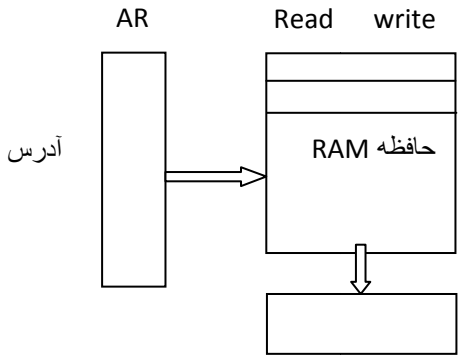
۴.۱) مدلهای قبلی توسعه یافته :

✓ **پردازش انجمنی (Associative Processing) :**

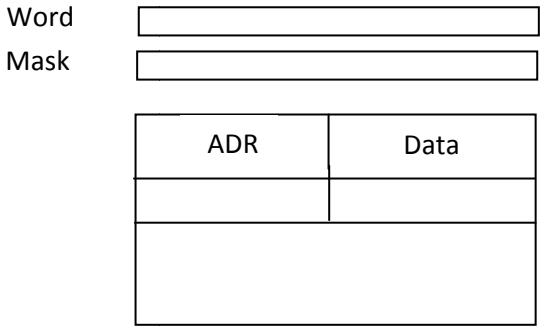
قابلیتها :

۱) جستجوی ماسک دار با جستجوی یک الگوی مشخص شده از تمامی کلمات حافظه بصورت موازی .

۲) نوشتن موازی یک الگوی بیتی در فیلد مشخص شده .



✓ **محتوای حافظه به عنوان آدرس Content Adress Memory (CAM) :**



Word بطور موازی با تمام کلمات حافظه مقایسه می شود که یا وجود دارد یا خیر (miss/hit)

شکل زیر دسته بندی مدلهای موازی را نشان می دهد :

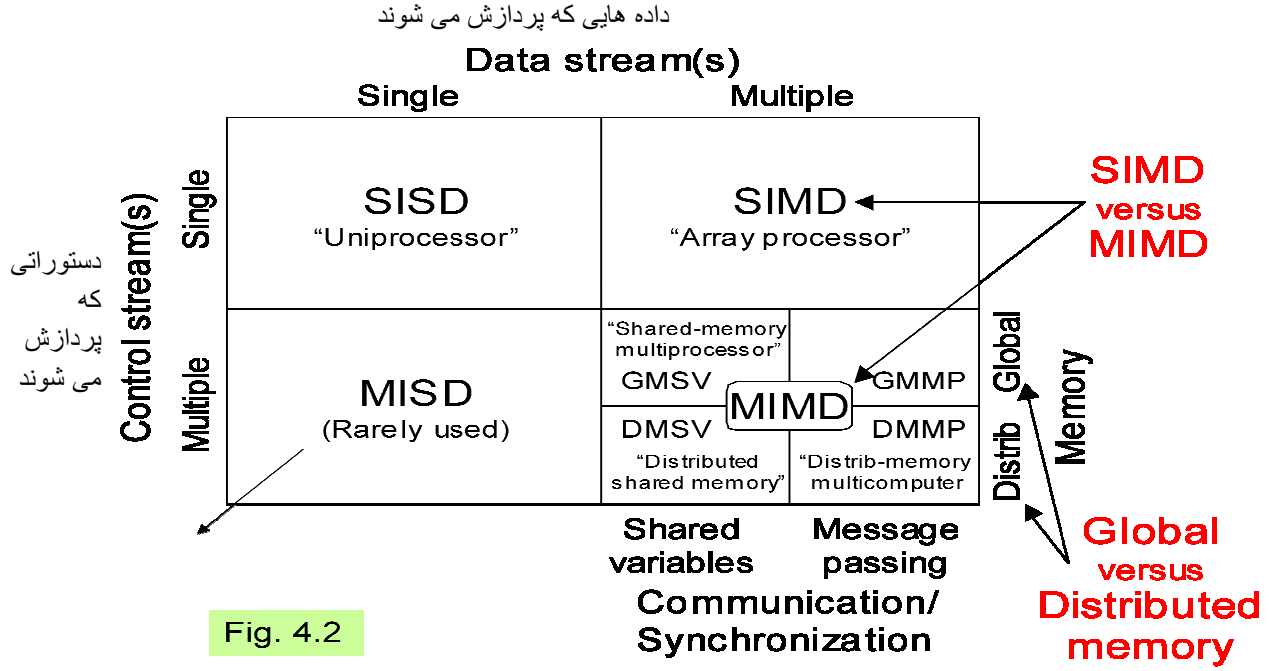


Fig. 4.2

۴.۲) مقایسه معماریهای SIMD و MIMD :

در اصل با این مقایسه می خواهیم بگوییم که چرا امروزه تمرکز روی MIMD می باشد.

معماری SIMD یک واحد کنترل دارد که دستورات را Fetch و دی کد نموده و سیگنال های کنترل را روی یک مجموعه داده برای اجرا پخش می کند.(در SIMD یک واحد کنترل برای همه پردازنده ها استفاده می شود ولی در MIMD هر پردازنده واحد کنترل خود را دارد)

مشخصات :

۱) اکثر معماریهای قدیمی ماشینهای موازی ، طراحی SIMD دارند.
۲) ماشینهای مدرن پردازنده های موازی MIMD می باشند.

۴.۳) مقایسه سیستمهای MIMD با حافظه های سرتاسری و توزیع شده :

✓ **معماری MIMD با حافظه سرتاسری :**

در معماری MIMD با حافظه ی سرتاسری تمامی پردازنده ها دسترسی یکنواخت به حافظه دارند و به آنها معماریهای UMA (Uniform Memory Access) نیز گفته می شود.

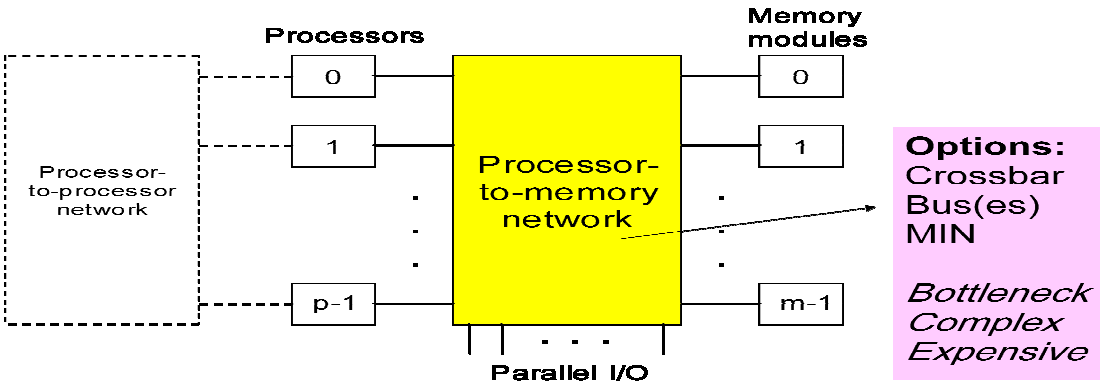


Fig. 4.3 A parallel processor with global memory.

اشکال این معماری بوجود آمدن گلوگاه روی شبکه (ترافیک زیاد) و راه حل آن افزودن حافظه Cache به پردازنده ها می باشد.

✓ **معماری MIMD با حافظه توزیع شده :**

در مدل حافظه های توزیع شده ، پیمانه های حافظه روی پردازنده ها به صورت مساوی پخش می شود و به آن NUMA(Noun Uniform Memory Access) می گویند.

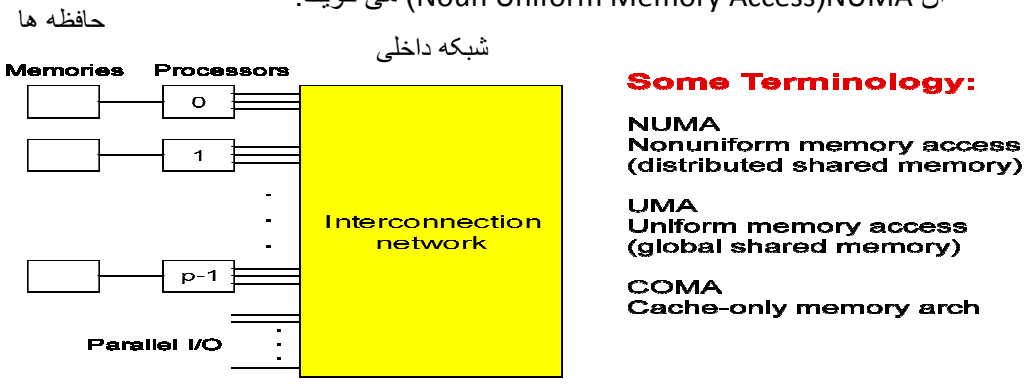


Fig. 4.5 A parallel processor with distributed memory.

۴.۴) مدل حافظه اشتراکی PRAM (parallel random-access machine):

در این مدل هر پردازنده می تواند به هر محل حافظه در هر سیکل زمان بدون توجه به اینکه پردازنده های دیگر چه می کنند دسترسی داشته باشد.

✓ **دیاگرام مدل حافظه اشتراکی PRAM در حالت تئوری (غیر واقعی):**

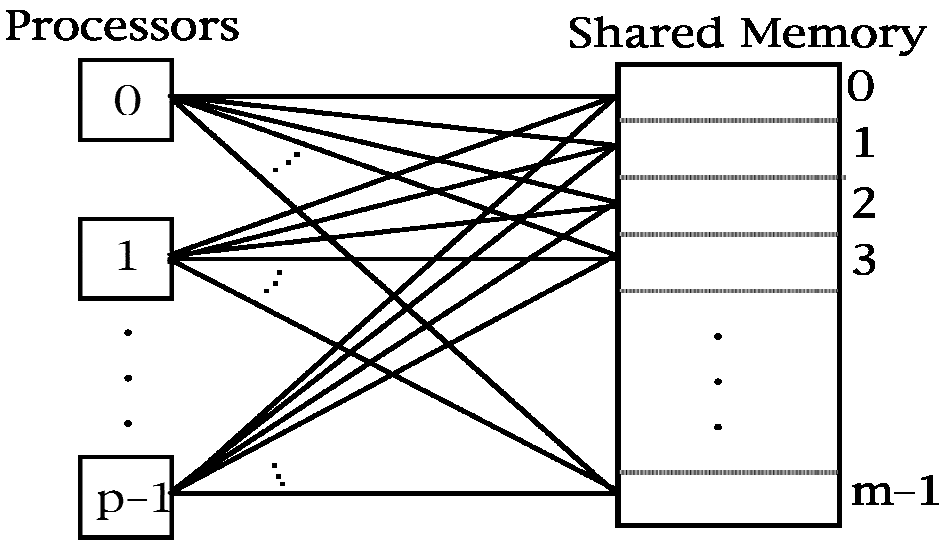


Fig. 4.6 Conceptual view of a parallel random-access machine (PRAM).

نکته : به هر خانه از حافظه در یک واحد زمانی بیش از یک پردازنده نمی تواند دسترسی داشته باشد. پس این دیاگرام درحالت تئوری است و درحالت عملی باید یک صف تشکیل شود که دردیگرام زیر نشان داده شده است.

✓ **دیاگرام مدل حافظه اشتراکی PRAM در حالت عملی (واقعی):**

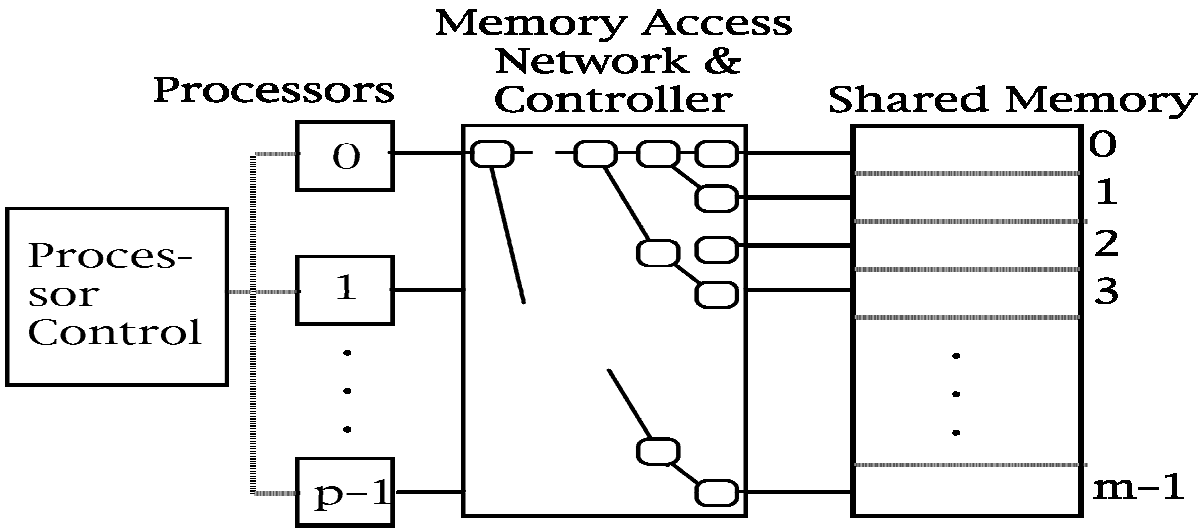


Fig. 4.7 PRAM with some hardware details shown.

دراین مدل هرپردازنده درخواست را به شبکه می دهد و این درخواست درصف قرارمی گیرد تا به حافظه مورد نظر دستیابی داشته باشد.

۴.۵) مدل های گراف برای حافظه های توزیع شده :

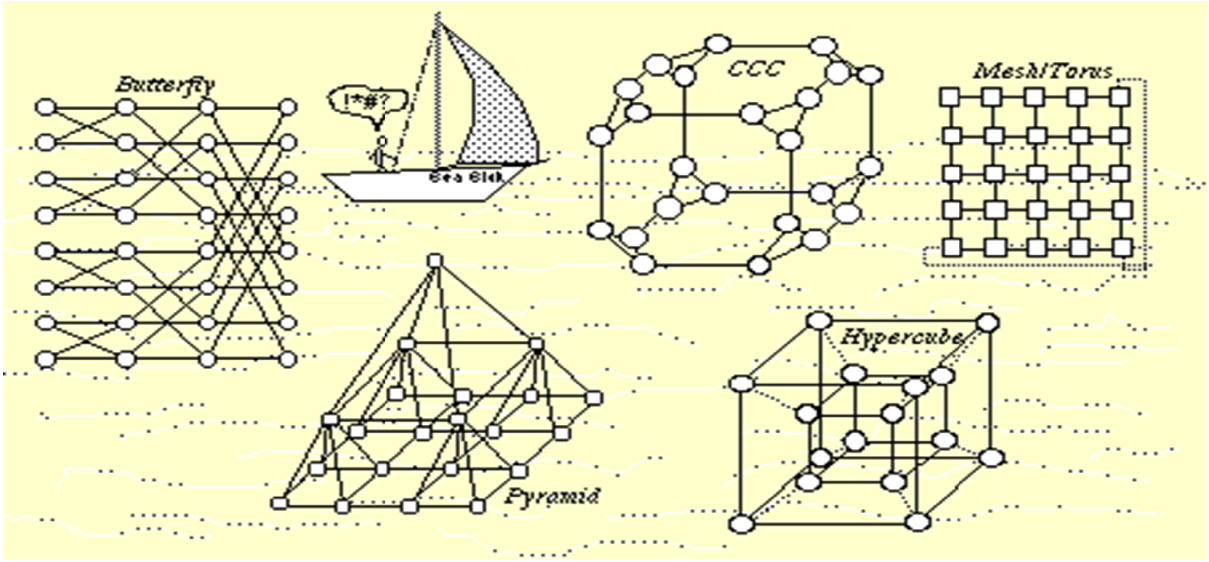


Fig. 4.8 The sea of interconnection networks.

پارامترهای مهم :

- ✓ **قطر شبکه (Network diameter):** طولانی ترین مسیرهای کوتاه بین زوج گره ها (پردازنده + حافظه)
- ✓ **Bisection band with :** کوچکترین (ظرفیت کل) تعداد پیوندها برای تقسیم یک شبکه به دو زیر شبکه با اندازه نصف.
- ✓ **درجه گره یا راس (Vertex Or Node degree) :** تعداد پورتهای ارتباطی مورد نیاز در هر گره .

جدول زیر پارامترهای فوق را بر روی برخی از شبکه های مدل گراف نشان می دهد:

Network name(s)	Number of nodes	Network diameter	Bisection width	Node degree	Local links?
1D mesh (linear array)	k	$k-1$	1	2	Yes
1D torus (ring, loop)	k	$k/2$	2	2	Yes
2D Mesh	k^2	$2k-2$	k	4	Yes
2D torus (k-ary 2-cube)	k^2	k	$2k$	4	Yes[1]
3D mesh	k^3	$3k-3$	k^2	6	Yes
3D torus (k-ary 3-cube)	k^3	$3k/2$	$2k^2$	6	Yes[1]
Pyramid	$(4k^2-1)/3$	$2\log_2 k$	$2k$	9	No
Binary tree	2^l-1	$2l-2$	1	3	No
4-ary hypertree	$2^l(2^{l+1}-1)$	$2l$	2^{l+1}	6	No
Butterfly	$2^l(l+1)$	$2l$	2^l	4	No
Hypercube	2^l	l	2^{l-1}	l	No
Cube-connected cycles	$2^l l$	$2l$	2^{l-1}	3	No
Shuffle-exchange	2^l	$2l-1$	$\geq 2^{l-1}/l$	4 unidir.	No
De Bruijn	2^l	l	$2^l/l$	4 unidir.	No

[1] With folded layout

قسمت دوم (مدلهای اضافی)

۵.۱. زیر ملهای PRAM و فرضیات :

هر پردازنده **i** در یک سیکل ۳ عمل زیر را انجام می دهد :

- ✓ عمل واکشی (Fetch) : واکشی مقدار از آدرس S_i در حافظه اشتراکی
- ✓ انجام برخی از محاسبات روی داده های قرار گرفته در رجیسترهای محلی
- ✓ ذخیره سازی یک مقدار در آدرس مقصد d_i در حافظه اشتراکی

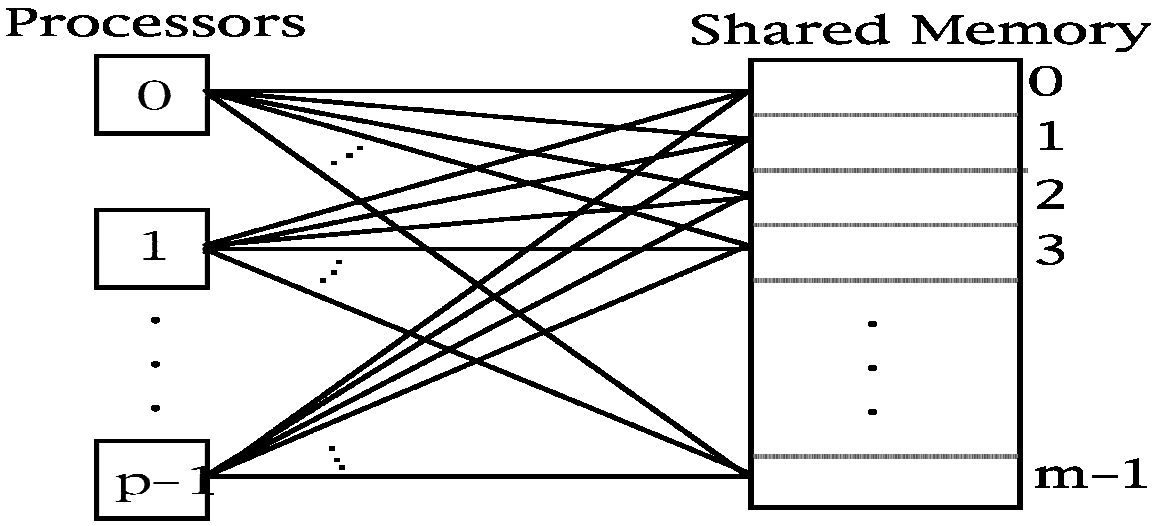


Fig. 4.6 Conceptual view of a parallel random-access machine (PRAM).

نکته ۱:دربرخی ازحالات ممکن است تمام مراحل فوق انجام نشده وفقط برخی ازمراحل انجام شود(بسته به دستور)

نکته ۲: آدرسهای S_i و d_i پردازنده P_i مستقل از پردازنده های دیگر است.

انواع (زیر مدلهای) PRAM :

بر مبنای ترتیب خواندن و نوشتن حافظه چهار زیر مدل تعریف می شود.

۱) EREW PRAM (**E***xclusive* **R***ead* **E***xclusive* **W***rite*)(خواندن و نوشتن انحصاری)
وقتی پردازنده ای خانه ای از حافظه را به منظور خواندن یا نوشتن در اختیار گرفت ، پردازنده های دیگر قادر به دسترسی به آن خانه از حافظه نمی باشند.

۲) ERCW PRAM (**E***xclusive* **R***ead* **C***oncurrent* **W***rite*)(خواندن انحصاری و نوشتن همزمان)
وقتی پردازنده ای خانه ای از حافظه را به منظور خواندن در اختیار گرفت ، پردازنده های دیگر قادر به دسترسی به آن خانه از حافظه نمی باشندولی پردازنده ها می توانند همزمان خانه ای از حافظه را بنویسند.
(این مدل عملا فاقد کارایی بوده و مفید نیست)

۳) CREW PRAM (**C***oncurrent* **R***ead* **E***xclusive* **W***rite*)(خواندن همزمان و نوشتن انحصاری)
وقتی پردازنده ای خانه ای از حافظه را به منظور خواندن در اختیار گرفت ، پردازنده های دیگر نیزقادر به خواندن ازآن خانه از حافظه می باشندولی پردازنده ها نمی توانند همزمان خانه ای از حافظه را بنویسند.

۴) CRCW PRAM (*Concurrent Read Concurrent Write*)(خواندن همزمان و نوشتن همزمان)
چند پردازنده می توانند همزمان از خانه ای از حافظه خوانده یا بنویسند. (بسیار قوی)

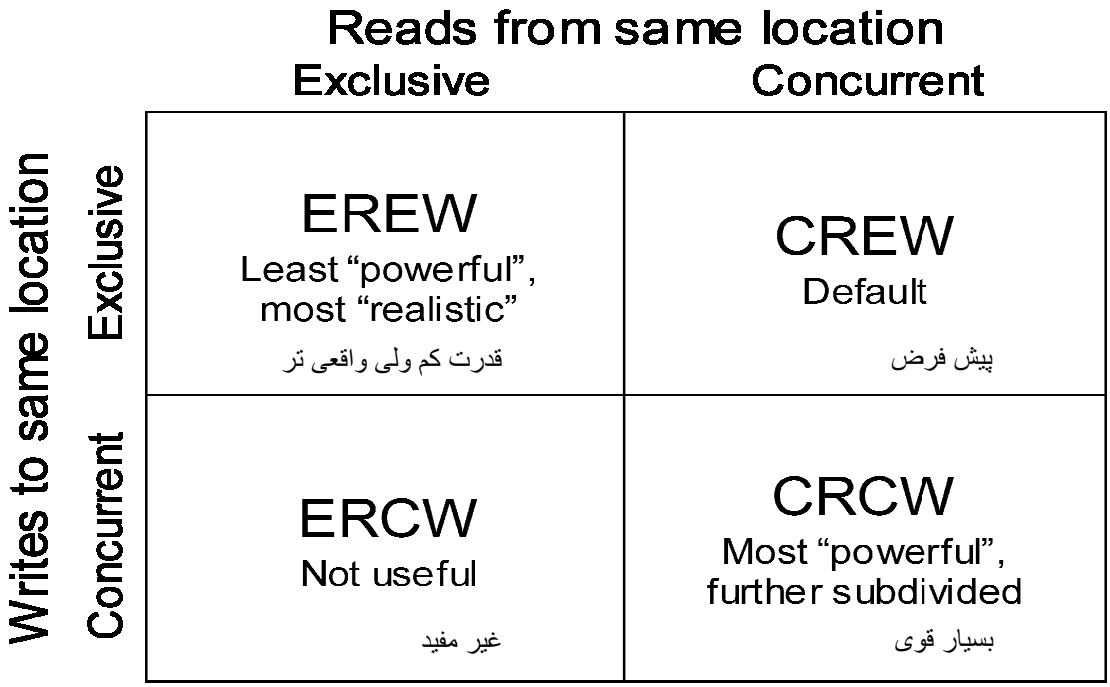


Fig. 5.1 Submodels of the PRAM model.

انواع CRCW PRAM :

بر اساس اینکه مدل CRCW نوشتن همزمان را چگونه انجام می دهد خود به تعدادی زیر مدل تقسیم می شود:

- ✓ **CRCW_U (U*ndifined*) (تعریف نشده):** مقدار نوشته شده نامشخص است .
- ✓ **CRCW_D (D*etecting*) (تشخیص برخورد بوسیله سخت افزار):** وقتی چند پردازنده در یک حافظه مشترک چند مقدار متفاوت بنویسند ، سخت افزار آنرا شناسایی کرده و اطلاع می دهد که "برخورد" بوجود آمده است.
- ✓ **CRCW_C (C*ommon*) (یکسان) :** اگر مقادیر نوشته شده توسط پردازنده های مختلف یکسان باشند عمل نوشتن مجاز است .(این زیر مدل نوشتن سازگار نامیده می شود)
- ✓ **CRCW_R (R*andom*) (تصادفی):** مقداری که قرار است دریک خانه از حافظه نوشته شود بطور تصادفی از پردازنده ها انتخاب می شود.
- ✓ **CRCW_P (P*riority*) (اولویت) :** پردازنده با کوچکترین شماره (اندیس)عمل نوشتن را انجام می دهد.
- ✓ **CRCW_M (M*ax/Min*) (بزرگترین یا کوچکترین) :** مقدار نوشته شده بر اساس بزرگترین یا کوچکترین مقدار تولید شده توسط چند پردازنده انتخاب می گردد.
- ✓ **CRCW_S (S*um*) (جمع) :** روی داده های تولید شده توسط چند پردازنده که قرار است در خانه ای از حافظه نوشته شود عمل "جمع" انجام شده و حاصلجمع در حافظه مورد نظر نوشته می گردد.
- ✓ **CRCW_A (A*nd*) ("AND"منطقی) :** روی داده های تولید شده توسط چند پردازنده که قرار است در خانه ای از حافظه نوشته شود عمل "AND" انجام شده و حاصل در حافظه مورد نظر نوشته می گردد.
- ✓ **CRCW_X (X*or*) ("XOR"منطقی) :**روی داده های تولیدشده توسط چند پردازنده که قراراست درخانه ای از حافظه نوشته شود عمل "XOR" انجام شده و حاصل در حافظه مورد نظر نوشته می گردد.

۵.۲. انتشار داده ها (DATA BRODCASTING) :

✓ **ساده ترین حالت انتشار One_To_All (انتشار داده از یک پردازنده به تمامی پردازنده های دیگر) در مدل EREW PRAM است. الگوریتم اصلی :**

با استفاده از این الگوریتم می خواهیم دیتا را از پردازنده P_0 به سایر پردازنده ها پخش کنیم .

Making P Copies of B[0] by Recursive Doubling

for $k = 0$ to $\lceil \log_2 p \rceil - 1$ Pj, $0 \le j < p$, do
Copy $B[j]$ into $B[j + 2^k]$
endfor

```
for k = 0 to ⌈log2 p⌉ – 1  do
   for  j =0 to  , p-1  do
      Copy Pj,  B[j] into B[j + 2^k]
   endfor
endfor
P: تعداد پردازنده ها        B: حافظه اشتراکی
```

مثال : یک سیستم دارای ۱۲ پردازنده بوده که از P_0 تا P_{11} شماره گذاری شده است . می خواهیم ببینیم بنا به روش "۲ برابر شدن بازگشتی" انتشار داده ها در مدل EREW چگونه انجام می شود

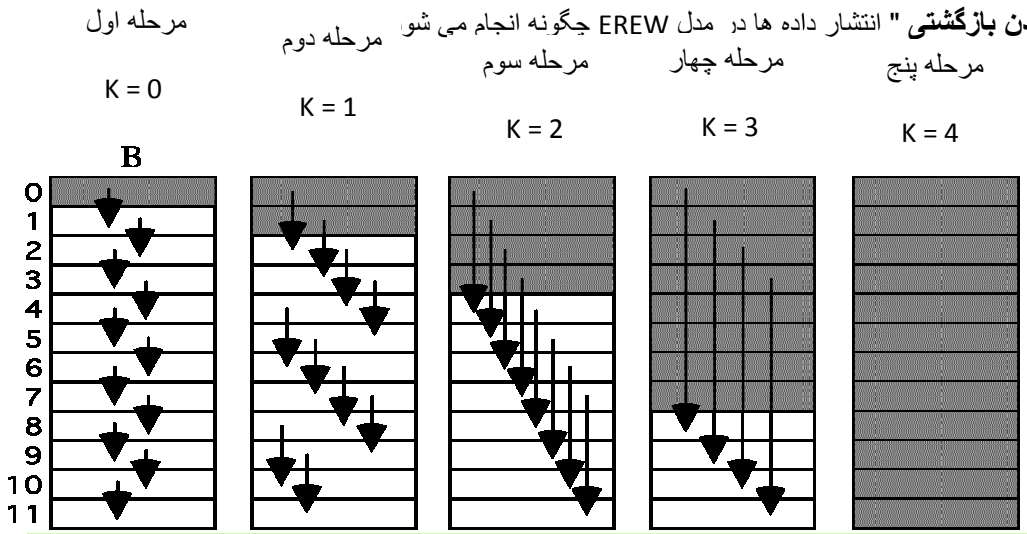


Fig. 5.2 Data broadcasting in EREW PRAM via recursive doubling.

همانطور که در دیاگرام فوق مشاهده می گردد در مرحله اول گردش الگوریتم (P_0 (K=0 مقدار خود (B[0]) را به P_1 منتشرکرده و P1 نیز مقدار خود (B[1]) که اکنون همان (B[0]) است به P2 منتشر کرده و ...(گام پرش ۱ است)

K	j	P_j	B[j]	$B[j+2^k]$
0	0	P0	B[0]	**B[1]**
0	1	P1	B[1]	**B[2]**
0	2	P2	B[2]	**B[3]**
0	3	P3	B[3]	**B[4]**
0	4	P4	B[4]	**B[5]**
0	5	P5	B[5]	**B[6]**
0	6	P6	B[6]	**B[7]**
0	7	P7	B[7]	**B[8]**
0	8	P8	B[8]	**B[9]**
0	9	P9	B[9]	**B[10]**
0	10	P10	B[10]	**B[11]**
0	11	P11		

همچنین در مرحله دوم گردش الگوریتم (K=1) P_0 مقدار خود (B[0]) را به P_2 منتشرکرده و P1 مقدار خود (B[1]) را به P3 منتشر کرده و ... (گام پرش ۲ است)

K	j	P_j	B[j]	$B[j+2^k]$
1	0	P0	B[0]	**B[2]**
1	1	P1	B[1]	**B[3]**
1	2	P2	B[2]	**B[4]**
1	3	P3	B[3]	**B[5]**
1	4	P4	B[4]	**B[6]**
1	5	P5	B[5]	**B[7]**
1	6	P6	B[6]	**B[8]**
1	7	P7	B[7]	**B[9]**
1	8	P8	B[8]	**B[10]**
1	9	P9	B[9]	**B[11]**
1	10	P10		
1	11	P11		

مرحله سوم (K=2): (گام پرش ۴ است)

K	j	P_j	B[j]	$B[j+2^k]$
2	0	P0	B[0]	**B[4]**
2	1	P1	B[1]	**B[5]**
2	2	P2	B[2]	**B[6]**
2	3	P3	B[3]	**B[7]**
2	4	P4	B[4]	**B[8]**
2	5	P5	B[5]	**B[9]**
2	6	P6	B[6]	**B[10]**
2	7	P7	B[7]	**B[11]**
2	8	P8		
2	9	P9		
2	10	P10		
2	11	P11		

مرحله چهارم (K=3): (گام پرش ۸ است)

K	j	P_j	B[j]	$B[j+2^k]$
3	0	P0	B[0]	**B[8]**
3	1	P1	B[1]	**B[9]**
3	2	P2	B[2]	**B[10]**
3	3	P3	B[3]	**B[11]**
3	4	P4		
3	5	P5		
3	6	P6		
3	7	P7		
3	8	P8		
3	9	P9		
3	10	P10		
3	11	P11		

مرحله پنجم (K=4): در این مرحله انتشار داده کامل شده است.

- اشکال این روش (الگوریتم) اینست که بسیاری از کپی ها بیهوده بوده و خیلی از کپی ها خارج از محدوده می باشد.

✓ **الگوریتم بهبود یافته انتشار داده ها در مدل EREW PRAM به روش "۲ برابر شدن بازگشتی" :**

به منظور رفع ۲ اشکال فوق الگوریتم زیر توین شده است:

P*i* write the data value into *B*[0]
***s* := 1**
while *s* < *p* **P*j*, 0 ≤ *j* < min(*s*, *p* – *s*), do**
Copy *B*[*j*] into *B*[*j* + *s*]
s* := 2*s
endwhile
P*j*, 0 ≤ *j* < *p*, read the data value in *B*[*j*]

```
while  s < p  do
  for  j =0 to  min(s, p – s)-1  do
     Copy Pj,  B[j] into B[j + s]
     s := 2s
  endfor
endwhile
```

در الگوریتم فوق شرط (S<P) به دلیل خارج نشدن کپی ها از محدوده تعداد پردازنده ها (P) می باشد

و نیز شرط **min(*S, p-S*)** به دلیل اجتناب از کپی های تکراری به آن افزوده شده است.

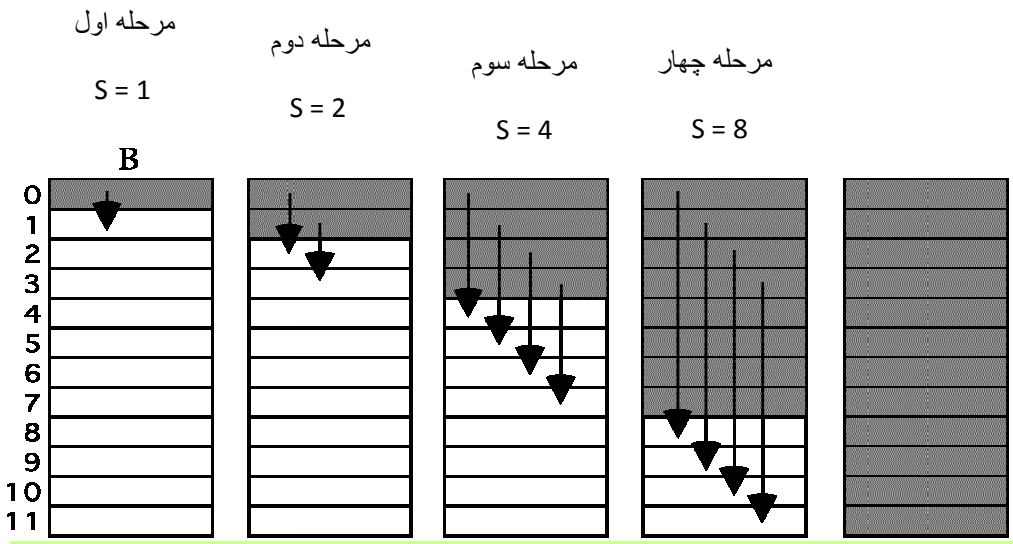


Fig. 5.3 EREW PRAM data broadcasting without redundant copying.

گردش اول (S=1): (گام پرش ۱ است)

S	j	P_j	Min(S,P-S)	B[j]	B[j+S]
1	0	P0	Min(1,12-1)=1	B[0]	B[1]

گردش دوم (S=2): (گام پرش ۲ است)

S	j	P_j	Min(S,P-S)	B[j]	B[j+S]
2	0	P0	Min(2,12-2)=2	B[0]	B[2]
2	1	P1	Min(2,12-2)=2	B[1]	B[3]

گردش سوم (S=4): (گام پرش ۴ است)

S	j	P_j	Min(S,P-S)	B[j]	B[j+S]
4	0	P0	Min(4,12-4)=4	B[0]	B[4]
4	1	P1	Min(4,12-4)=4	B[1]	B[5]
4	2	P2	Min(4,12-4)=4	B[2]	B[6]
4	3	P3	Min(4,12-4)=4	B[3]	B[7]

گردش چهارم (S=8): (گام پرش ۸ است)

S	j	P_j	Min(S,P-S)	B[j]	B[j+S]
8	0	P0	Min(8,12-8)=4	B[0]	**B[8]**
8	1	P1	Min(8,12-8)=4	B[1]	**B[9]**
8	2	P2	Min(8,12-8)=4	B[2]	**B[10]**
8	3	P3	Min(8,12-8)=4	B[3]	**B[11]**

گردش پنجم (S=16): چون (S>P) یا (16>12) پس الگوریتم خاتمه می پذیرد.

❖ **پیچیدگی زمانی الگوریتم بهبود یافته $\theta(\log P)$ می باشد.**

✓ **انتشار All_To_All** (انتشار داده از همه پردازنده ها به تمامی پردازنده های دیگر)**درمدل EREW PRAM است.** (در این حالت هر پردازنده یک مقدار به (P-1) پردازنده دیگر ارسال می کند.)

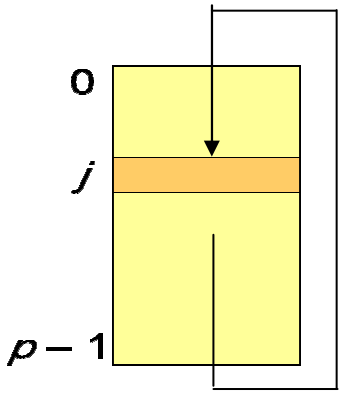
مدل ساده : **Processor *j*, $0 \le j < p$, write own data value into *B*[*j*]**

ابتدا همه پردازنده ها (P0 تا P_{p-1}) داده خود را در حافظه مخصوص به خود در B می نویسند.وسپس

for *k* = 1 to *p* – 1 Processor *j*, $0 \le j < p$, do
Read the data value in *B*[(*j* + *k*) mod *p*]
endfor

هرپردازنده همه داده های پردازنده های دیگر را می خواند.

(خواندن مقادیر پردازنده های دیگر بصورت حلقوی است)



❖ **یکی از کاربردهای الگوریتم فوق مرتب سازی (SORT) داده ها می باشد.**

الگوریتم مرتب سازی آرایه با انتشار ALL_TO_ALL :

هرخانه از حافظه مربوط به یک پردازنده بوده که می خواهیم آنها را بصورت حبابی مرتب کنیم :

Processor *j*, $0 \le j < p$, write 0 into *R*[*j*]
for *k* = 1 to *p* – 1 Processor *j*, $0 \le j < p$, do
$l := (j + k) \bmod p$
if *S*[*l*] < *S*[*j*] or *S*[*l*] = *S*[*j*] and *l* < *j*
then *R*[*j*] := *R*[*j*] + 1
endif
endfor
Processor *j*, $0 \le j < p$, write *S* [*j*] into *S*[*R*[*j*]]

ابتدا آرایه ***R*** که مربوط به رتبه داده هر پردازنده می باشد صفر می گردد. سپس بطور موازی پردازنده ها ۲ خانه ***j*** ام و ***l*** ام از آرایه ***S*** را با هم مقایسه کرده (با علامت کوچکتر) و در صورت برقراری شرط به پردازنده مربوطه (***j*** ام) اندیس خود را در آرایه رتبه (***R***) یک واحد افزایش می دهد.

پس از اتمام الگوریتم آرایه ***R*** رتبه هر پردازنده را مشخص کرده است .

P : تعداد پردازنده ها

S[j] : آرایه ای که داده مربوط به هر پردازنده در اندیس متناظر با شماره آن پردازنده قرار دارد.

R[j] : رتبه مربوط به داده هر پردازنده که در ***S*** موجود است.

مثال : مطلوبست مرور الگوریتم فوق برای مرتب سازی داده ها در ۴ پردازنده : (P=4)

	S
0	7
1	12
2	2
3	12

	R
0	0
1	0
2	0
3	0

گردش اول :(K=1)

K	j	L=(j+k) mod P	Pj	$S[j]$	$S[l]$	$R[j]$
1	0	1	P0	7	12	**0**
1	1	2	P1	12	2	**1**
1	2	3	P2	2	12	**0**
1	3	0	P3	12	7	**1**

گردش دوم :(K=2)

K	j	L=(j+k) mod P	Pj	$S[j]$	$S[l]$	$R[j]$
2	0	2	P0	7	2	**1**
2	1	3	P1	12	12	**1**
2	2	0	P2	2	7	**0**
2	3	1	P3	12	12	**2**

S[j]=s[l] ولی چون L>j پس R[3] اضافه می شود

گردش سوم :(K=3)

K	j	L=(j+k) mod P	Pj	$S[j]$	$S[l]$	$R[j]$
3	0	3	P0	7	12	**1**
3	1	0	P1	12	7	**2**
3	2	1	P2	2	12	**0**
3	3	2	P3	12	2	**3**

S[j]=s[l] ولی چون L>j پس R[3] اضافه می شود

وبنا به خط آخر الگوریتم بر اساس Rank بدست آمده عناصر S مرتب می گردند.

Processor *j*, $0 \le j < p$, write $S[j]$ into $S[R[j]]$

	S
0	2
1	7
2	12
3	12

۵.۳. محاسبات Semigroup :

بهترین سخت افزار برای محاسبات Semigroup سخت افزار CRCW از نوع مدلهای CRCW_S, CRCW_A , CRCW_X و ... می باشد.

در ادامه الگوریتم Semigroup را بر روی ماشینهای EREW PRAM بررسی می کنیم :

EREW PRAM semigroup computation algorithm

Proc j, $0 \leq j < p$, copy $X[j]$ into $S[j]$
$s := 1$
while $s < p$ Proc j, $0 \leq j < p - s$, do
    $S[j + s] := S[j] \otimes S[j + s]$
    $s := 2s$
endwhile
Broadcast $S[p - 1]$ to all processors

این عملگر میتواند جمع یا تفریق یا ... باشد

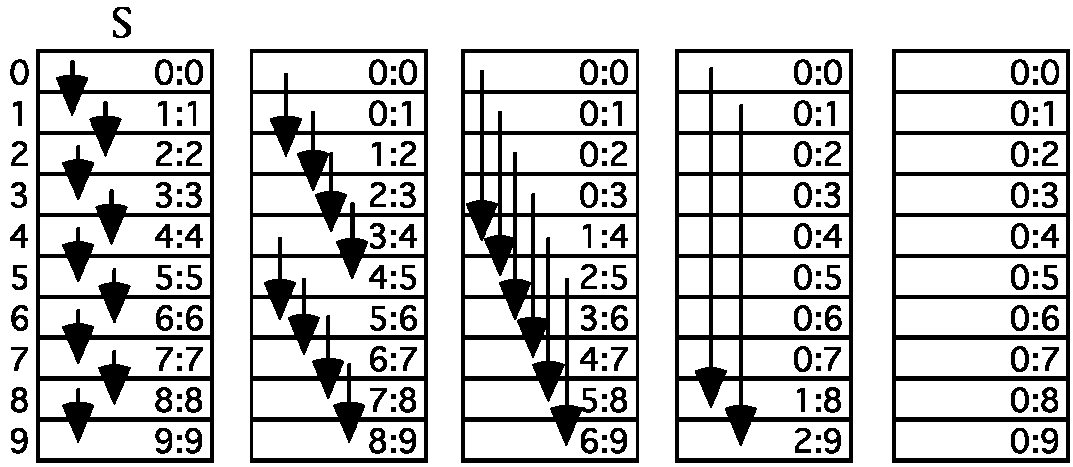


Fig. 5.4 Semigroup computation in EREW PRAM.

Same as the first part of semigroup computation (no final broadcasting)

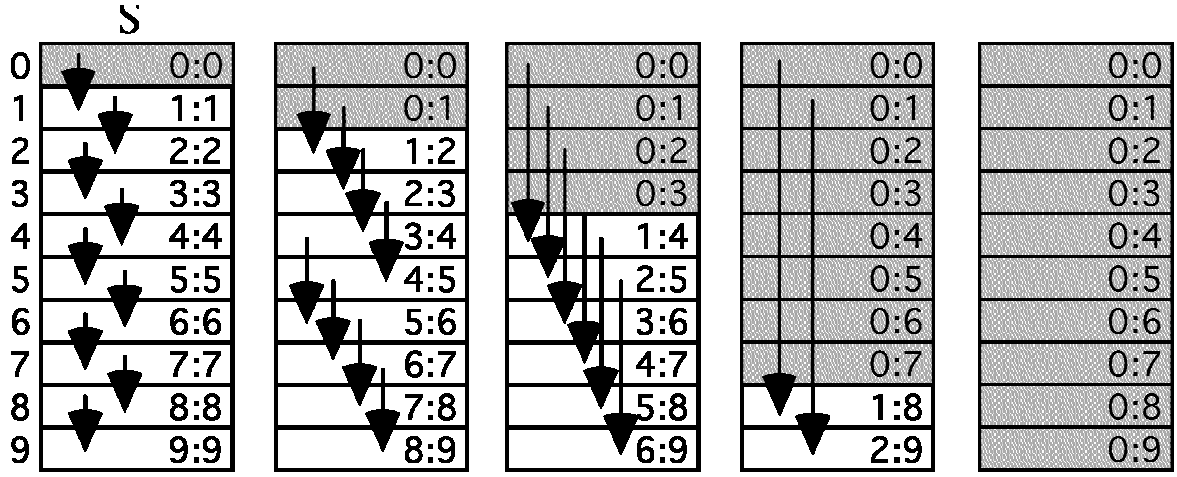


Fig. 5.6 Parallel prefix computation in EREW PRAM via recursive doubling.

حال به بیان چند الگوریتم در این ضمینه می پردازیم :

- ✓ الگوریتم تقسیم و غلبه به روش Parallel Orefix (حالت اولیه):

A Divide-and-Conquer Parallel-Prefix Algorithm

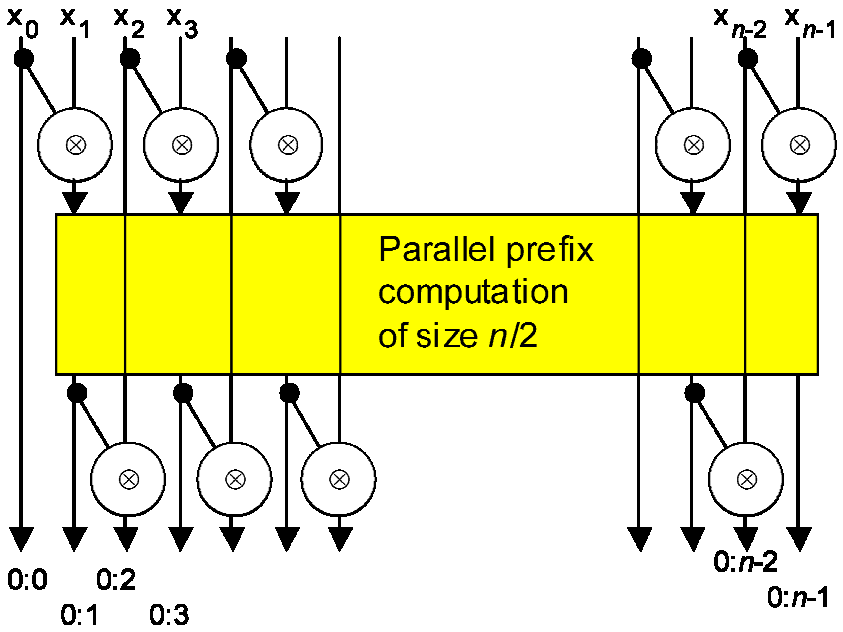


Fig. 5.7 Parallel prefix computation using a divide-and-conquer scheme.

✓ **الگوریتم تقسیم و غلبه به روش Parallel Orefix (حالت ثانویه):**

در این حالت اگر n تا پردازنده داریم و آنها را به ۲ تا n/2 تقسیم می کنیم تا سرعت بیشتر شود (ورودیهای زوج جدا و وردیهای فرد نیز جدا پردازش شده و درنهایت نتایج ترکیب می شوند).

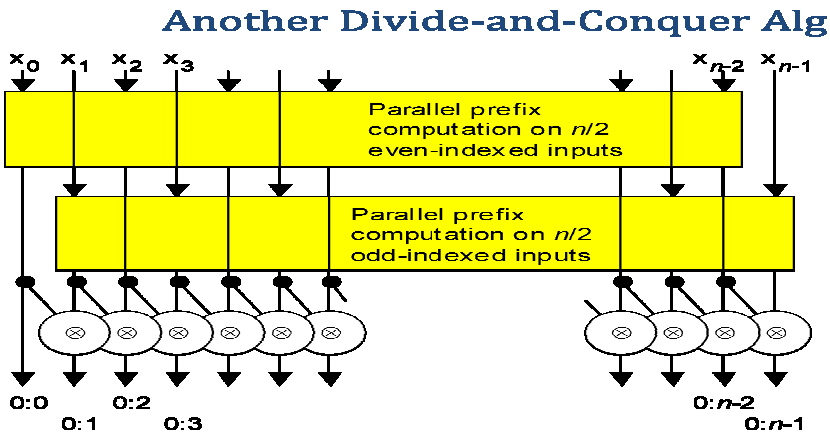


Fig. 5.8 Another divide-and-conquer scheme for parallel prefix computation.

۵.۵. رتبه بندی عناصر یک لیست پیوندی(Ranking the Elements of a Linked List) :

رتبه بندی (تعیین جایگاه هر نود در لیست)عناصر یک لیست پیوندی به این صورت انجام می پذیرد که :

- ✓ یک لیست پیوندی خطی با P عنصر داریم
- ✓ ۲ رنک می زنیم (یکی فاصله هر گره تا گره پایانی و دیگری تا ابتدای لیست)
- ✓ پیچیدگی معمولی (ترتیبی)این الگوریتم$\Theta(P)$ است که برای کم کردن پیچیدگی از PRAM استفاده می کنیم.

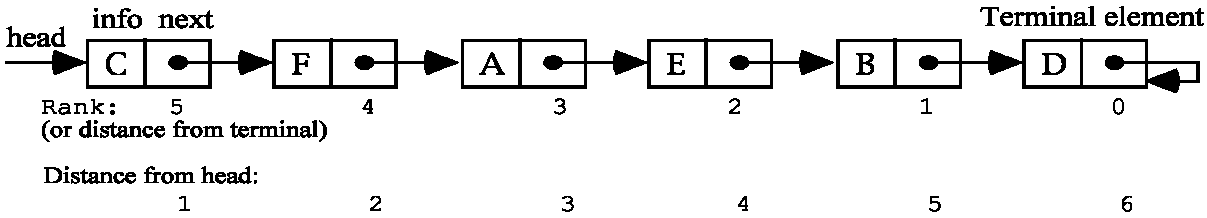


Fig. 5.9 Example linked list and the ranks of its elements.

پیاده سازی رتبه بندی فوق با PRAM: یک ساختمان داده به تعداد نودهای لیست پیوندی درست می کنیم :

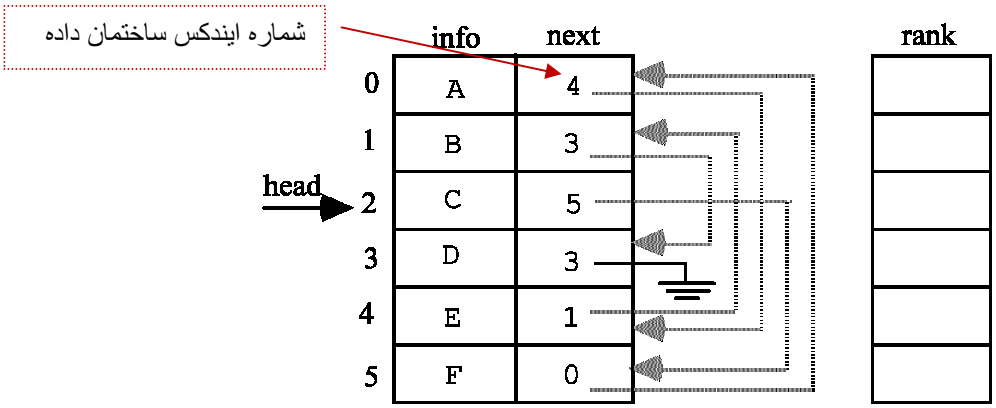


Fig. 5.10 PRAM data structures representing a linked list and the ranking results.

ساختمان داده فوق (آرایه) شامل ۲ ستون info و next می باشد که محتوای ستون Next ایندیس ساختمان داده ای است که با توجه به شکل 5.9 یک نود به نود دیگر اشاره می کند (مثلا چون نود A در شکل 5.9 به نود E اشاره می کند و نود E در ساختمان داده دارای اندیس ۴ می باشد لذا محتوای Next نود A در ساختمان داده ۴ است)

حال به بیان الگوریتم رتبه دهی (Rank) در PRAM با استفاده از ساختمان داده فوق می پردازیم :

PRAM list ranking algorithm (via pointer jumping)
Processor *j*, $0 \le j < p$, do {initialize the partial ranks}
if *next*[*j*] = *j*
then *rank*[*j*] := 0
else *rank*[*j*] := 1
endif
while *rank*[*next*[*head*]] $\overline{\neq}$0 Processor *j*, $0 \le j < p$, do
***rank*[*j*] := *rank*[*j*] + *rank* [*next*[*j*]]**
***next*[*j*] := *next*[*next*[*j*]]**
endwhile

مرور الگوریتم :

بر اساس قسمت If الگوریتم فوق ابتدا آرایه Rank مقدار اولیه می پذیرد:

	RANK
0	1
1	1
2	1
3	0
4	1
5	1

سپس بر اساس تکرار قسمت While خواهیم داشت :

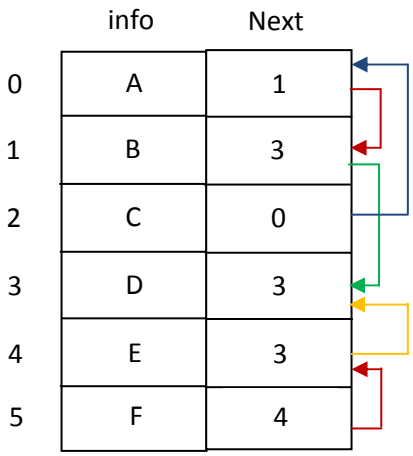
تکرار اول :

الف : تغییر Rank توسط عبارت rank[j]:=rank[j]+rank[next[j]]

RANK

2
1
2
0
2
2

ب: تغییر next توسط عبارت Next[j]:=Next[Next[j]]

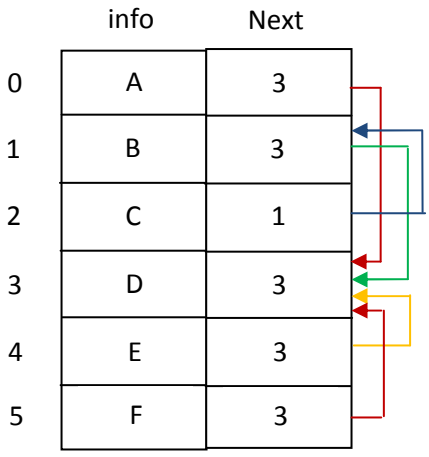


تکرار دوم :

الف : تغییر Rank توسط عبارت rank[j]:=rank[j]+rank[next[j]]

RANK
3
1
4
0
2
4

ب: تغییر next توسط عبارت Next[j]:=Next[Next[j]]



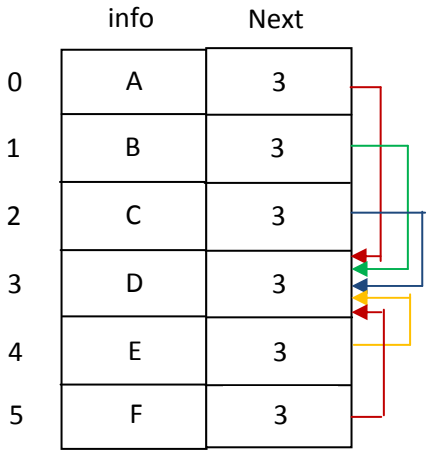
تکرار سوم :

الف : تغییر Rank توسط عبارت rank[j]:=rank[j]+rank[next[j]]

RANK

3
1
5
0
2
4

ب: تغییر next توسط عبارت Next[j]:=Next[Next[j]]



حال الگوریتم پایان یافته و فاصله هر نود از گره پایانی مشخص شده است (طبق آرایه Rank)

- پیچیدگی این الگوریتم $\Theta(n/2)$ می باشد.(سرعت نسبت به حالت ترتیبی تقریبا ۲ برابر شده است)

سئوال : کدام مدل از PRAM برای این الگوریتم مناسب است ؟ **(جواب :** مدل CREW)

۵.۶. ضرب ماتریسها (Matrix Multiplication) : (برای ماتریسهای مربع M X M)

همانطور که می دانید در ضرب ماتریسها ، ماتریس حاصلضرب از حاصل جمع ضرب سطردر ستون بدست می آید.

C=A x B

$$C_{ij} = \sum_{k=0}^{m-1} a_{ik} \times$$

پیچیدگی الگوریتم ضرب فوق در حالت ترتیبی $\Theta(m^3)$ می باشد. ولی اگر بصورت ترتیبی PRAM انجام شود از پیچیدگی کمتری برخوردار است.(که آنرا در ۴ حالت مختلف بررسی خواهیم کرد (بسته به تعداد پردازنده ها))

حالت اول : PRAM دارای $P=M^3$ پردازنده باشد.

این حالت همان حالت ترتیبی است با این تفاوت که عمل جمع موازی انجام می شود :

پیچیدگی الگوریتم در این حالت $\Theta(\log m)$ می باشد.

Sequential matrix multiplication

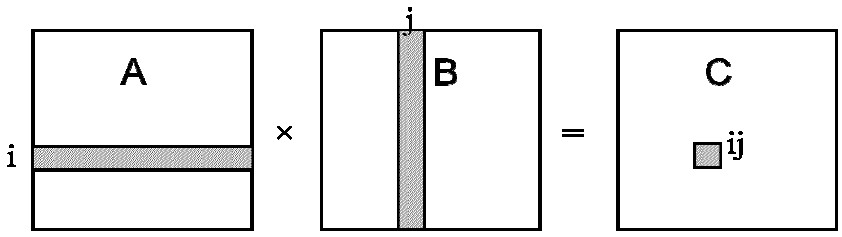
```
for i = 0 to m – 1 do
    for j = 0 to m – 1 do
        t := 0
        for k = 0 to m – 1 do
            t := t + a_ik b_kj
        endfor
        c_ij := t
    endfor
endfor
```

PRAM solution with m^3 processors: each processor does one multiplication (not very efficient)

$$c_{ij} := \sum_{k=0 \text{ to } m-1} a_{ik} b_{kj}$$

A B C

$m \times m$ matrices



حالت دوم : PRAM دارای $P=M^2$ پردازنده باشد. پیچیدگی الگوریتم در این حالت برابر است با $\Theta(m)$:

- ✓ در این حالت هر پردازنده یک عنصر C_{ij} را بدست می آورد.

PRAM matrix multiplication using m^2 processors

```
Proc (i, j), 0 ≤ i, j < m, do
begin
    t := 0
    for k = 0 to m – 1 do
        t := t + a_ik b_kj
    endfor
    c_ij := t
end
```

- ✓ به علت اینکه چند پردازنده یک ردیف یکسان از A را با یک ردیف یکسان از B می خوانند ، پیاده سازی این الگوریتم با مدل CREW می باشد.

اشکال این الگوریتم اینست که باز هم تعداد پردازنده ها زیاد است.

حالت سوم : PRAM دارای P=M پردازنده باشد. پیچیدگی الگوریتم در این حالت برابر است با $\Theta(m^2)$:

PRAM matrix multiplication using *m* processors

```
for j = 0 to m – 1 Proc i, 0 ≤ i < m, do
    t := 0
    for k = 0 to m – 1 do
        t := t + a_ik b_kj
    endfor
    c_ij := t
endfor
```

- ✓ به علت اینکه هر پردازنده یک سطر متفاوت از ماتریس A را می خواند ، خواندن همزمان A نیاز نیست ولی ماتریس B باید همزمان خوانده شود.(خواندن موازی برای ستونهای B لازم است)

حالت چهارم : PRAM دارای P<M پردازنده باشد.

در این حالت ماتریس اصلی را به یکسری زیر ماتریس تقسیم می کنند(بسته به تعداد پردازنده ها) ، سپس ضربها را روی زیر ماتریسها انجام داده و سپس آنها را با هم جمع می کنند.

۶.الگوریتمهای بیشتر برای معماری حافظه اشتراکی (Shared-Memory)

در این فصل به بررسی الگوریتمهای پیچیده تری نسبت به الگوریتمهای فصل ۵ خواهیم پرداخت .

۶.۱. انتخاب مبتنی برپایه رتبه ترتیبی(Sequential Ranked-Based Selection)

- ✓ هدف از این الگوریتم پیدا نمودن K امین عنصر کوچک از یک دنباله ($S=X_0, X_1,\ldots, X_{n-1}$)می باشد.
- ✓ موارد استفاده این الگوریتم پیدا کردن میانه ، ماکزیمم ، مینیمم و ... در یک دنباله می باشد.
- ✓ الگوریتم ترتیبی دارای پیچیدگی O(n log n) بوده در صورتی که الگوریتم موازی آن دارای پیچیدگی O(n) است.

الگوریتم بهبود یافته ترتیبی :

Linear-Time Sequential Selection Algorithm

Sequential rank-based selection algorithm *select*(*S*, *k*)

1. if $|S| < q$ {q is a small constant}
   then sort S and return the kth smallest element of S
   else divide S into $|S|/q$ subsequences of size q
   Sort each subsequence and find its median
   Let the $|S|/q$ medians form the sequence T
   endif

$O(n)$

2. m = *select*(T, $|T|/2$) {find the median m of the $|S|/q$ medians}

$T(n/q)$

3. Create 3 subsequences
   L: Elements of S that are $< m$
   E: Elements of S that are $= m$
   G: Elements of S that are $> m$

$O(n)$

4. if $|L| \geq k$
   then return *select*(L, k)
   else if $|L| + |E| \geq k$
   then return m
   else return *select*(G, $k - |L| - |E|$)
   endif
   endif

$T(3n/4)$

S : لیست عناصر |S| : طول لیست عناصر

K : k امین عدد به لحاظ کوچکی

q : یک ثابت کوچک که عناصر لیست به آن تقسیم میشوند.(برای تکه تکه کردن عناصر لیست)

این الگوریتم در ۴ مرحله پیاده سازی می گردد:

۱. تقسیم کردن عناصر لیست به تعداد ($|S|/q$) و مرتب سازی (سورت) هر قسمت
۲. پیدا کردن عنصر میانه هر زیر مجموعه لیست (m) و انتقال این میانه ها به آرایه T
۳. ایجاد ۳ زیر مجموعه از لیست (S)که :
   - ✓ L : عناصری از S که از میانه m کوچکترند.
   - ✓ E : عناصری از S که با میانه m برابرند.
   - ✓ G : عناصری از S که از میانه m بزرگترند.
۴. جستجوی بازگشتی برای یافتن k امین عنصر

مثال : برای لیست زیر (S) به طول ۲۵ (n=25) و (q=5) داده ها با رتبه ی K=5 و K=9 و K=13 را پیدا کنید.

S= 6 4 5 6 7 1 5 3 8 2 1 0 3 4 5 6 2 1 7 1 4 5 4 9 5

حل : بند ۱ الگوریتم :

$|S|$=25 و q=5 چون $|S| > q$ پس مجموعه S به زیر مجموعه های 5 عنصری تقسیم می گردند($|S|/q$).

S= 6 4 5 6 7 1 5 3 8 2 1 0 3 4 5 6 2 1 7 1 4 5 4 9 5

سپس عناصر هر زیر مجموعه مرتب (سورت) می گردند:

S= 4 5 6 6 7 1 2 3 5 8 0 1 3 4 5 1 1 2 6 7 4 4 5 5 9

سپس عنصر میانه هر زیر مجموعه تعیین و در آرایه T قرار می گرند:

T= 6 3 3 2 5

بند ۲ الگوریتم : سپس میانه ، میانه ها را بدست می آوریم(m) :

M = 3

بند ۳ الگوریتم : حال ۳ زیر مجموعه S با توجه به مقدار m محاسبه شده در مرحله قبل را بدست می آوریم :

L = 1 2 1 0 2 1 1 ($|L|$=7) {عناصری ازS که از m کوچکترند }

E = 3 3 ($|E|$=2) {عناصری ازS که با m برابرند }

G = 6 4 5 6 7 5 8 4 5 6 7 4 5 4 9 5 ($|G|$=16) {عناصری ازS که از m بزرگترند }

بند ۴ الگوریتم : چون الف (k=5) و ($|L|$=7) بنا شرط $|L| \geq k$ لذا عملیات فوق را مجددا با در نظر گرفتن S=L و q=k بصورت بازگشتی تکرار می کنیم

بند ۱ الگوریتم : $|S|$=7 و q=5 چون $|S|$>q پس مجموعه S به زیرمجموعه های 5عنصری تقسیم می گردند($|S|/q$).

S = 1 2 1 0 2 1 1

سپس عناصر هر زیر مجموعه مرتب (سورت) می گردند:

S = 0 1 1 2 2 1 1

سپس عنصر میانه هر زیر مجموعه تعیین و در آرایه T قرار می گرند:

T= 1 1

بند ۲ الگوریتم : سپس میانه ، میانه ها را بدست می آوریم(m) :

M = 1

بند ۳ الگوریتم : حال ۳ زیر مجموعه S با توجه به مقدار m محاسبه شده در مرحله قبل را بدست می آوریم :

L = 0 (|L|=1) {عناصری ازS که از m کوچکترند }

E = 1 1 1 1 (|E|=4) {عناصری ازS که با m برابرند }

G = 2 2 (|G|=2) {عناصری ازS که از m بزرگترند }

بند ۴ الگوریتم : چون الف (k=5) و (|L|=1) بنا شرط $|L| + |E| \geq k$ لذا مقدار m=1 بازگردانده می شود.

نتیجه اینکه پنجمین عنصر(K=5) لیست اصلی (S) عدد یک (m=1) می باشد.

قسمت (ب) مسئله : یافتن داده های لیست با رتبه K=9 :

برای یافتن نهمین عنصر(K=9) لیست اصلی (s) بندهای ۱ تا ۳ انجام شده و (m=3) آمده و ۳ مجموعه L ، E و G تعیین می گردند که (|L|=7) و (|E|=2) و (|G|=16) می باشد .

بنا به بند ۴ الگوریتم چون $|L| + |E| \geq k$ پس m=3 تعیین می گردد. (یعنی نهمین عنصر لیست S عدد ۳ می باشد)

قسمت (ج) مسئله : یافتن داده های لیست با رتبه K=13 :

برای یافتن سیزدهمین عنصر(K=13) لیست اصلی (s) بندهای ۱ تا ۳ انجام شده و (m=3) آمده و ۳ مجموعه L ، E و G تعیین می گردند که (|L|=7) و (|E|=2) و (|G|=16) می باشد .

بنا به بند ۴ الگوریتم چون $|L| + |E| < k$ پس S=G و q=k- |L| - |E| تعیین و مجددا الگوریتم فراخوانی می گردد.

بند ۱ الگوریتم: |S|=16 و q=4 چون |S|>q پس مجموعه S به زیرمجموعه های ۴عنصری تقسیم می گردد($|S|/q$).

S = 6 4 5 6 7 5 8 4 5 6 7 4 5 4 9 5

سپس عناصر هر زیر مجموعه مرتب (سورت) می گردند:

S = 4 5 6 6 4 5 7 8 4 5 6 7 4 5 5 9

سپس عنصر میانه هر زیر مجموعه تعیین و در آرایه T قرار می گرند:

T= 5 5 5 5

بند ۲ الگوریتم : سپس میانه ، میانه ها را بدست می آوریم(m) :

M = 5

بند ۳ الگوریتم : حال ۳ زیر مجموعه S با توجه به مقدار m محاسبه شده در مرحله قبل را بدست می آوریم :

L = 4 4 4 4 (|L|=4) {عناصری ازS که از m کوچکترند }

E = 5 5 5 5 5 (|E|=5) {عناصری ازS که با m برابرند }

G = 6 6 7 8 6 7 9 (|G|=7) {عناصری ازS که از m بزرگترند }

بند ۴ الگوریتم : چون الف (k=4) و (|L|=4) بنا شرط $|L| \geq k$ لذا عملیات فوق را مجددا با در نظر گرفتن S=L و q=k بصورت بازگشتی تکرار می کنیم لذا مقدار m=4 بازگردانده می شود.

۶.۲. الگوریتم انتخاب موازی (A Parallel Selection Algorithm)

الگوریتم (۱) : مرتب سازی آرایه و انتخاب K امین عنصر آرایه :

- ✓ در این الگوریتم از مدل CRCW PRAM-S که S حرف اول کلمه SUM می باشد استفاده می شود .
- ✓ در این مدل از تعداد $P=n^2$ (n تعداد عناصر) پردازنده استفاده می شود.
- ✓ این الگوریتم دارای ۳ سیکل است :

سیکل اول :

- ✓ هر پردازنده دارای اندیس **(I,j)** می باشد که $0 \le i,j \le n$
- ✓ پردازنده **(I,j)** خانه های آرایه **S[i]** و **S[j]** را مقایسه می کند و در صورتیکه **S[j]<S[i]** (نزولی) **یا S[i]=S[j],i<j** باشد مقدار **1** را در خانه **RANK[j]** حافظه می نویسد.
- ✓ چون از خاصیت جمع نوشتن های همزمانی استفاده می شود ، بعد از این سیکل **RANK[j]** ، رتبه **S[j]** مرتب شده در لیست را نشان می دهد.

سیکل دوم :

- ✓ در این سیکل پردازنده $(0,j)$, $0 \le j < n$ ، **S[j]** را خوانده ودر $S[rank[j]]$ می نویسد.

سیکل سوم :

- ✓ فرایند انتخاب (Selection) هنگامی کامل میگردد که خانه $S[k-1]$ توسط همه پردازنده ها خوانده شود (K کوچکترین عنصر آرایه S می باشد)

مثال) فرض کنید آرایه S شامل 5 عنصر نامرتب زیر می باشد که می خواهیم این عناصر را بر اساس الگوریتم فوق مرتب کنیم (n=5)

	S	RANK
0	7	0
1	12	0
2	5	0
3	16	0
4	3	0

طبق سیکل یک الگوریتم پردازنده **(I,j)** خانه های آرایه **S[i]** و **S[j]** را مقایسه می کند و در صورتیکه **S[j]<S[i]** **(نزولی) یا S[i]=S[j],i<j** باشد مقدار **1** را در خانه **RANK[j]** حافظه می نویسد.

مقایسه خانه 0 با 4 خانه دیگر توسط پردازنده ها	(0,1) ، (0,2) ، (0,3) ، (0,4)
آیا شرط برقرار است یا خیر	0 1 0 1
مقایسه خانه 1 با 4 خانه دیگر توسط پردازنده ها	(1,0) ، (1,2) ، (1,3) ، (1,4)
آیا شرط برقرار است یا خیر	1 1 0 1
مقایسه خانه 2 با 4 خانه دیگر توسط پردازنده ها	(2,0) ، (2,1) ، (2,3) ، (2,4)
آیا شرط برقرار است یا خیر	0 0 0 1

مقایسه خانه 3 با 4 خانه دیگر توسط پردازنده ها (3,4) ، (3,2) ، (3,1) ، (3,0)

آیا شرط برقرار است یا خیر 1 1 1 1

مقایسه خانه 4 با 4 خانه دیگر توسط پردازنده ها (4,3) ، (4,2) ، (4,1) ، (4,0)

آیا شرط برقرار است یا خیر 0 0 0 0

بنابراین آرایه Rank به قرار زیر تغییر می کند :

خانه 4	خانه 3	خانه 2	خانه 1	خانه 0	وضعیت اولیه	مقایسه خانه (I,j)
2	2	1	1	0	0	خانه 0
1	1	0	0	0	0	خانه 1
3	3	2	2	1	0	خانه 2
0	0	0	0	0	0	خانه 3
4	4	3	2	1	0	خانه 4

بنا بر این وضعیت نهایی آرایه Rank بصورت فوق بوده که در سیکل ۲ الگوریتم آرایه S بر اساس آرایه Rank مرتب می گردد($S\ [rank[j]]$).

	S
0	3
1	5
2	7
3	12
4	16

نتیجه گیری :

- ✓ در سیکل اول از n^2 پردازنده استفاده می شود.
- ✓ در سیکل دوم از **n** پردازنده استفاده می شود.
- ✓ در سیکل سوم از **1** پردازنده استفاده می شود .

الگوریتم (۲) : مرتب سازی آرایه و انتخاب K امین عنصر آرایه(نسخه موازی) :

Parallel rank-based selection algorithm *PRAMselect(S, k, p)*
1. if |*S* | < 4
then sort *S* and return the *k*th smallest element of *S*
else broadcast |*S*| to all *p* processors
divide *S* into *p* subsequences *S*(*j*) of size |*S* | /*p*
Processor *j*, 0 ≤ *j* < *p*, compute the median *Tj* : = *select* (*S(j)*, |*S*(*j*)|/2)
endif
2. *m* = *PRAMselect* (*T*, |*T*|/2, *p*) {find the median of the medians in parallel}
3. Broadcast *m* to all processors and create 3 subsequences
L*: Elements of *S* that are < *m
E*: Elements of *S* that are = *m
G*: Elements of *S* that are > *m
4. if |*L*| ≥ *k*
then return *PRAMselect* (*L, k, p*)
else if |*L*| + |*E*| ≥ *k*
then return *m*
else return *PRAMselect* (*G, k* – |*L* | – |*E*|, *p*)
endif

۶.۳. الگوریتم مرتب سازی بر مبنای انتخاب (A SELECTION-BASED SORTING ALGORITHM)

Parallel selection-based sorting algorithm *PRAMselectionsort(S, p)*
1. if |*S*| < *k* then return *quicksort* (*S*)
2. for *i* = 1 to *k* – 1 do
***mi* := *PRAMselect* (*S, i* |*S*|/*k, p*)**
{for notational convenience, let *mo* := –∞ ; *mk* := +∞ }
endfor
3. for *i* = 0 to *k* – 1 do
make the sublist *T (i)* from elements of *S* that are between *mi* and *mi*+1
endfor
4. for *i* = 1 to *k* /2 do in parallel
***PRAMselectionsort* (*T (i)*, 2*p*/*k*)**
{*p*/(*k*/2) processors are used for each of the *k*/2 subproblems}
endfor
5. for *i* = *k*/2 + 1 to *k* do in parallel
***PRAMselectionsort*(*T (i)*, 2*p*/*k*)**
Endfor

که در این الگوریتم S آرایه و P تعداد پردازنده ها می باشد.

حال الگوریتم را با یک مثال شرح می دهیم :

مثال : آرایه ای را با اندازه ۲۵ عنصر (|S|=25) و ۵ پردازنده (P=5) در نظر بگیرید .(آستانه K برابر است با $k=2^{1/x}$ اگر x=1/2 آنگاه K=4 خواهد بود) مطلوبست اجرای الگوریتم فوق :

S: 6 4 5 6 7 1 5 3 8 2 1 0 3 4 5 6 2 1 7 0 4 5 4 9 5

چون |S| > *k* (۲۵>۴) در این الگوریتم از آستانه k ام جهت تقسیم بندی لیست به K=4 زیر لیست استفاده می کنیم :

2. for *i* = 1 to *3* do
***mi* := *PRAMselect* (*S, i* 25/*4, 5*)**
{for notational convenience, let *mo* := –∞ ; *mk* := +∞ }
Endfor

i=1 ⟹ 1*25/4 ≈ 6 m_1= ***PRAMselect* (*S,6, 5*)** = 2

i=2 ⟹ 2*25/4 ≈ 13 m_2= ***PRAMselect* (*S,13, 5*)** = 4

i=3 ⟹ 3*25/4 ≈ 19 m_3= ***PRAMselect* (*S,19, 5*)** = 6

که همواره ∞+ **:= *m*₄ ;** ∞– **:= m_0** می باشد . حال طبق m های بدست آمده آرایه S را به ۴ زیر لیست تقسیم بندی می کنیم :

3. for *i* = 0 to *k* – 1 do
make the sublist *T (i)* from elements of *S* that are between *mi* and *mi*+1
endfor

i=0 ⟹ T_0=**{0,0,1,1,1,2}** عناصر بین m_0 و m_1

i=1 ⟹ T_1=**{2,3,3,4,4,4,4}** عناصر بین m_1 و m_2

i=2 ⟹ T_2=**{5,5,5,5,5,6}** عناصر بین m_2 و m_3

i=3 ⟹ T_3=**{6,6,7,7,8,9}** عناصر بین m_3 و m_4

***T* : - - - - - 2|- - - - - - 4|- - - - - 6|- - - - - -**

***T* : 0 0 1 1 1 2|2 3 3 4 4 4 4|5 5 5 5 5 6|6 6 7 7 8 9**

حال طبق بند ۴ و ۵ الگوریتم مجددا الگوریتم برای ۴ زیر مجموعه فوق اجرا می گردد(توجه شود که در هر مرحله تعداد پردازنده ها کم می گردد)

4. for *i* = 1 to *k* /2 do in parallel
***PRAMselectionsort* (*T (i)*, 2*p/k*)**
{*p*/(*k*/2) processors are used for each of the *k*/2 subproblems}
endfor
5. for *i* = *k*/2 + 1 to *k* do in parallel
***PRAMselectionsort*(*T (i)*, 2*p/k*)**
Endfor

این الگوریتم آنقدر اجرا می گردد (مجموعه ها هر بار کوچکتر می شوند) تا عناصر S کاملا مرتب شوند.

فصل ۷ : شبکه های مرتب سازی و انتخاب (Sorting and Selection Networks)

۷.۱) شبکه مرتب سازی چیست ؟

یک شبکه مرتب سازی مداری است که n ورودی $x_0,x_1,x_2,\ldots,x_{n-1}$ را دریافت نموده و با جایگزینی آنها ، n خروجی $y_0,y_1,y_2,\ldots,y_{n-1}$ تولید می کند بطوریکه $y_0 \le y_1 \le y_2 \le \ldots \le y_{n-1}$ باشد.

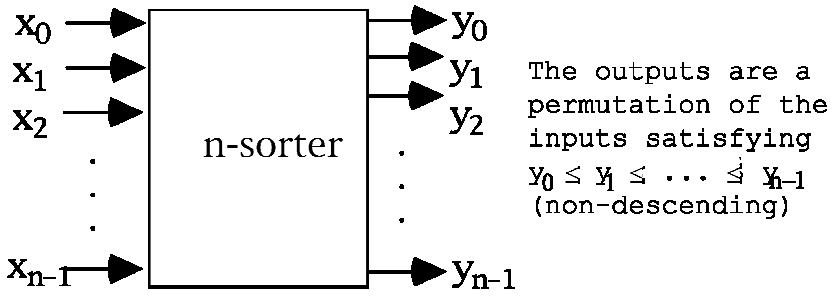


Fig. 7.1 An *n*-input sorting network or an *n*-sorter.

بلوک دیاگرام شبکه مرتب سازی ۲ بیتی (2_sorter) :

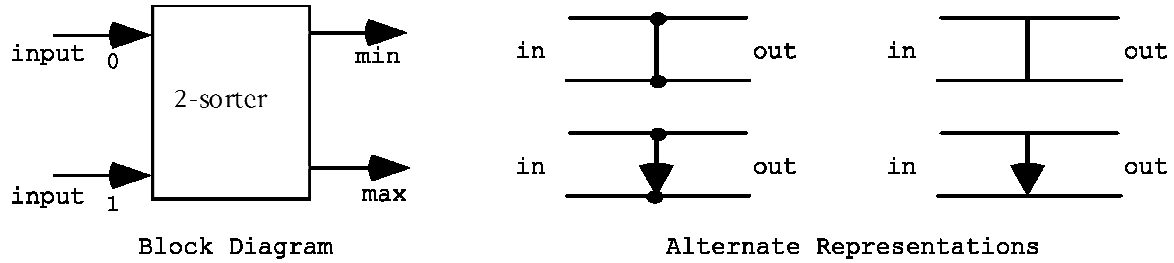


Fig. 7.2 Block diagram and four different schematic representations for a 2-sorter.

یک 2_sorter به این صورت کار می کند که مقدار کم (minimom) را در خروجی بالا (y_0) و مقدار زیاد (Maximom) را در خروجی پایین (y_1) ظاهر می کند.

بلوک دیاگرام سخت افزاری 2_sorter : (هر ورودی میتواند شامل K بیت باشد**)**

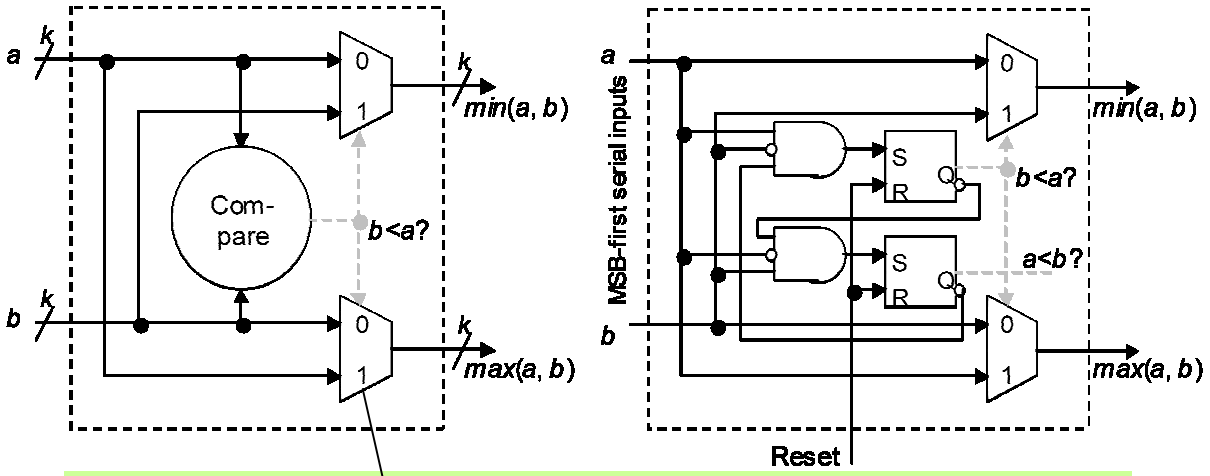


Fig. 7.3 Parallel and bit-serial hardware realizations of a 2-sorter.

مالتی پلکسر

پیاده سازی بصورت بیتهای ورودی موازی

پیاده سازی بصورت بیتهای ورودی سریال

اگر a>b باشد ورودی های صفر انتخاب می شوند و در غیر اینصورت ورودی های یک انتخاب می گردند.

شکل زیر یک 4_sorter را به همراه یک مثال با استفاده از بلوکهای 2_sorter نشان می دهد :

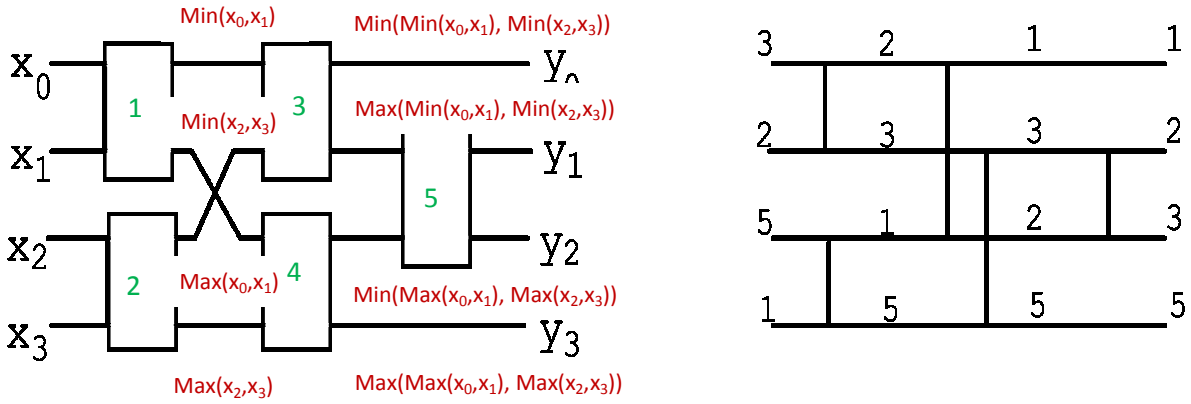


Fig. 7.4 Block diagram and schematic representation of a 4-sorter.

4_sorter فوق از ۵ عدد 2_sorter تشکیل شده است و عملکرد آنها بدین صورت می باشد که همزمان X0 و X1 توسط سورتر شماره ۱و X2 و X3 توسط سورتر شماره ۲ مرتب میگردند بدین صورت که مقدار کوچکتر بین ورودی های X0 و X1 به خروجی بالا و مقدار بزرگتر بین ورودی های X0 و X1 به خروجی پایین سورتر شماره ۱ منتقل می گردد(چنین است برای سورتر شماره ۲) حال سورتر شماره ۳ Min(X0,X1) و Min(X2,X3) را به عنوان وردی دریافت کرده و مقدار کمینه آنها (Min(Min(X2,X3) , Min(X0,X1))) را در خروجی بالا و مقدار بیشینه آنها را در خروجی پایین ظاهر می کند (Max(Min(X2,X3) , Min(X0,X1))) و چنین است برای سورتر شماره ۴. این روند در سورتر شماره ۵ نیز ادامه می یابد تا تمامی ۴ عنصر مرتب گردند.

۷.۲. پارامترهای شایستگی و کارایی شبکه های مرتب سازی :

شبکه های مرتب سازی (n_sorter) دارای دو پارامتر زیر می باشند که هر چه مقدار این دو پارامتر کم باشد شبکه از کارایی بیشتری برخوردار است .

- ✓ **هزینه (Cost) :** تعداد کل مقایسه گرها (2_sorter) استفاده شده در طراحی شبکه می باشد.
- ✓ **تاخیر(Delay) :** تعداد سطوح (تعداد کل 2_sorter روی یک مسیر بحرانی از ورودی به خروجی می باشد)
  - **مسیر بحرانی :** مسیری که از ورودی به خروجی بیشترین تعداد بلوک 2_sorter را داشته باشد.

حاصلضرب هزینه در تاخیر ، میزان کارایی شبکه را نشان می دهد(Cost * Delay)

در مثال فوق Cost=5 و Delay=3 بوده که حاصلضرب آنها برابر 15 می باشد.

مثالهایی از کارایی و شایستگی در طراحی های (با هزینه کم و تاخیر ثابت) مشخص است :

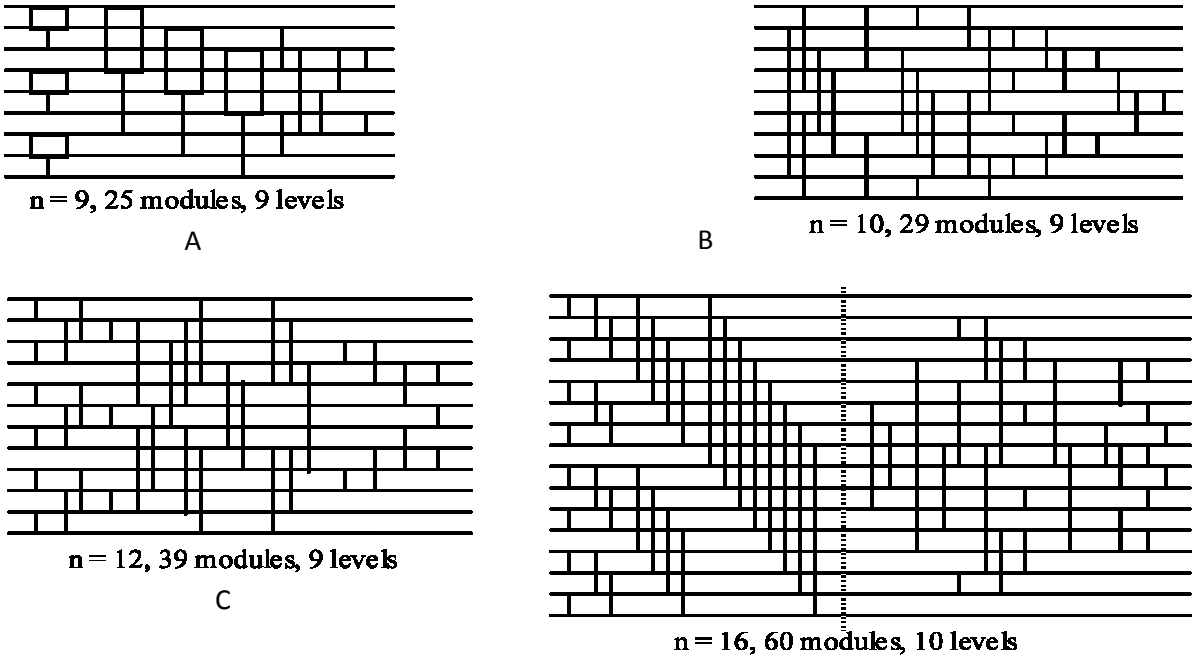


Fig. 7.5 Some low-cost sorting networks.

A) شبکه مرتب سازی با ۹ وردی (n=9) و هزینه ۲۵ (Cost=25) و تعداد سطح ۹ (Delay=9)
Cost X Delay=25 X 9 =225

B) شبکه مرتب سازی با ۱۰ وردی (n=10) و هزینه ۲۹ (Cost=29) و تعداد سطح ۹ (Delay=9)
Cost X Delay=29 X 9 =261

C) شبکه مرتب سازی با ۱۲ وردی (n=12) و هزینه ۳۹ (Cost=39) و تعداد سطح ۹ (Delay=9)
Cost X Delay=39 X 9 =351

D) شبکه مرتب سازی با ۱۶ وردی (n=16) و هزینه ۶۰ (Cost=60) و تعداد سطح ۱۰ (Delay=10)
Cost X Delay=60 X 10 =600

مثالهایی از کارایی و شایستگی در طراحی های (مرتب سازی سریع) مشخص است :

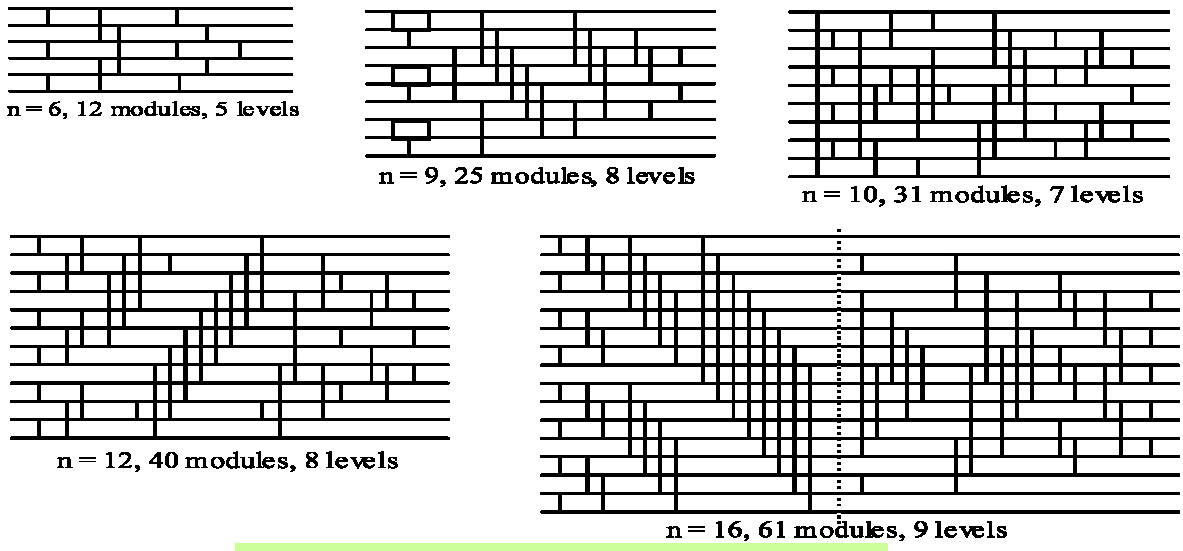


Fig. 7.6 Some fast sorting networks.

۷.۳. طراحی شبکه های مرتب سازی :

۱) روش مرتب سازی با تاخیر و تقدم زوج و فرد (Odd-Even-Transposition) :

یک روش طراحی شبکه مرتب سازی روش مرتب سازی با تاخیر و تقدم زوج و فرد **(Odd-Even-transposition)** است که به آن طراحی دیوارآجری نیزمی گویند.(الگوریتم مرتب سازی فوق مربوط به بخش۲.۳ است که برای آرایه های خطی پردازنده ها شرح داده شد)

از مزایای این روش آسانی در سیم کشی می باشد. ولی بدلیل برخورداری از **n(n-1)/2** پیمانه (تعداد 2_sorter) و تاخیر (**n**) از کارایی پایینی (Cost * Delay=**n * n(n-1)/2= n^3**) برخوردار است ((**$\Theta(n^3)$**)

$$
\begin{aligned}
C(n) &= n(n-1)/2 \\
D(n) &= n \\
\text{Cost} \times \text{Delay} &= n^2(n-1)/2 = \Theta(n^3)
\end{aligned}
$$

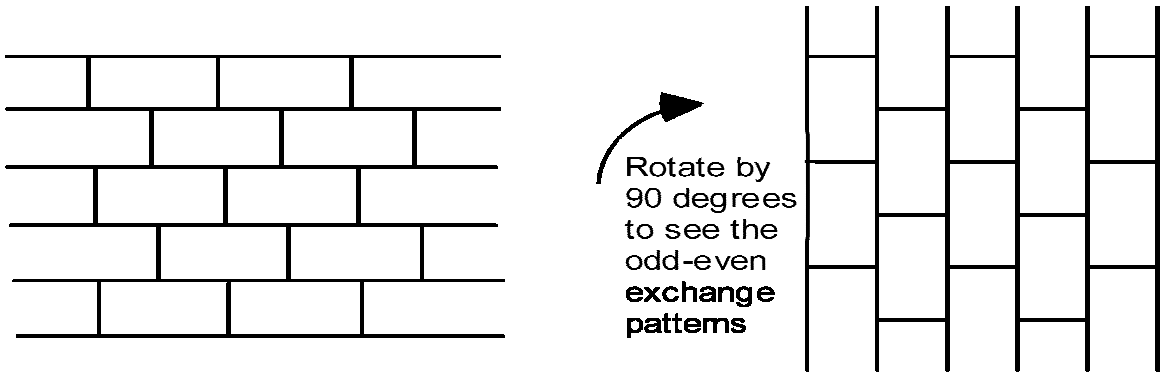
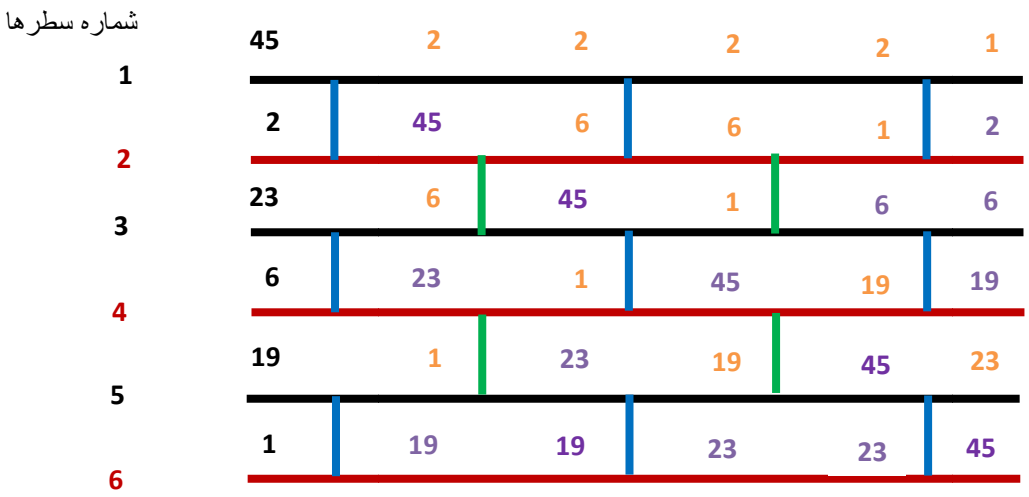


Fig. 7.7 Brick-wall 6-sorter based on odd–even transposition.

مثال : طراحی یک شبکه مرتب سازی 6_sorter با روش Odd-Even :



خطوط سیاه رنگ سطرهای فرد و خطوط قرمز رنگ سطرهای زوج می باشند.

در این مدل درسطح یک عمل مقایسه بین سطرهای فرد و زوج صورت می گیرد(۳ مقایسه) . سپس در سطح ۲ عمل مقایسه بین سطرهای زوج و فرد صورت می گیرد(۲ مقایسه) . درسطح ۳ عمل مقایسه بین سطرهای فرد و زوج صورت می گیرد(۳ مقایسه) . سپس در سطح ۴ عمل مقایسه بین سطرهای زوج و فرد صورت می گیرد(۲ مقایسه) .و در نهایت درسطح ۵ عمل مقایسه بین سطرهای فرد و زوج صورت می گیرد(۳ مقایسه) .بدین ترتیب با (n-1=5) سطح مقایسه اطلاعات مرتب می گردند.

۳) روش مرتب سازی براساس درج و مرتب سازی بر اساس انتخاب (Insertion Sort and Selection Sort) :

۳.۱) روش مرتب سازی براساس درج (Insertion Sort):

در این روش به منظور مرتب سازی n ورودی ، در ابتدا n-1 ورودی اول را مرتب کرده و سپس آخرین ورودی را در محل مناسب درج می کنیم . (این الگوریتم از ۳ وردی آغاز و بصورت بازگشتی تا n داده را مرتب می کند. یعنی ابتدا ۲ ورودی را سورت کرده و سپس ورودی سوم را در محل مناسب درج می کند.حال که ۳ ورودی سورت است ورودی چهارم را درمکان مناسب درج می کند و... این عملیات را بصورت بازگشتی تا n وردی انجام می دهد)

۳.۲) روش مرتب سازی براساس انتخاب (Selection sort) :

در این روش بزرگترین مقدار از n مقداروروی انتخاب شده و در آخرین خروجی قرار می گیرد سپس از میان n-1 مقدار باقی مانده بزرگترین عدد انتخاب و در خط آخر قرار می گیرد . این روش بصورت بازگشتی تا زمانی انجام می گردد تا همه مقادیر مرتب گردند.

۳.۲) روش مرتب سازی بر اساس ترکیب روشهای فوق (Insertion sort and Selection Sort):

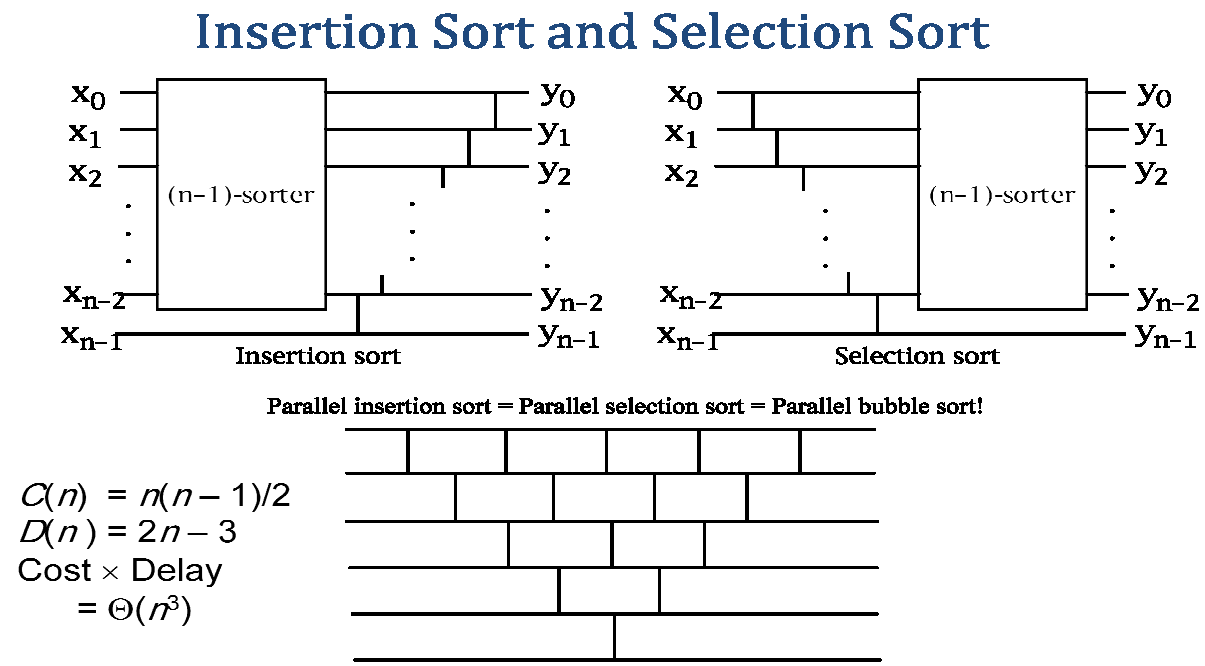
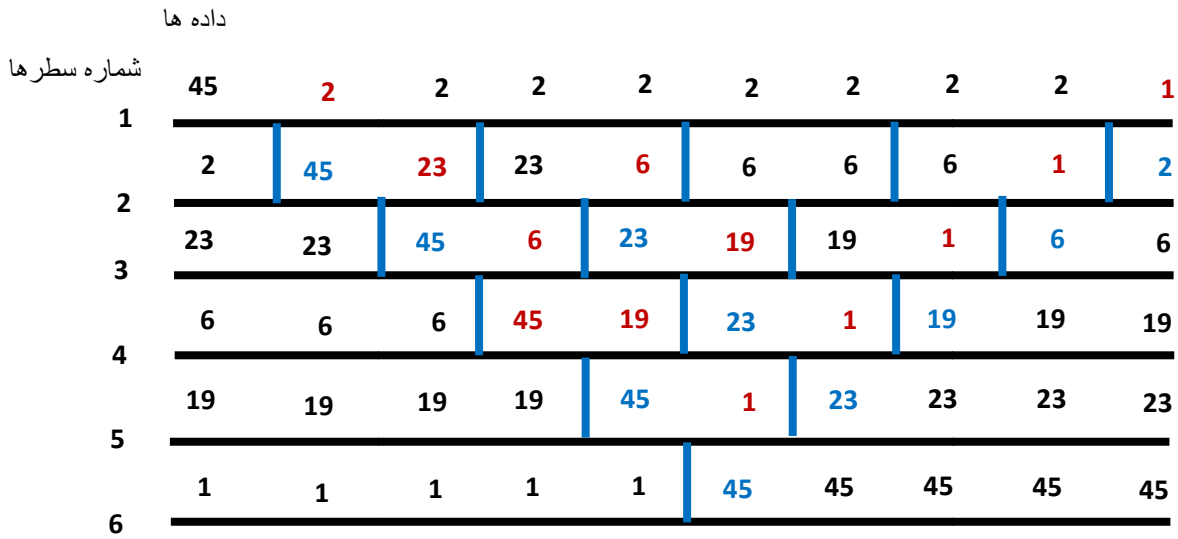


Fig. 7.8 Sorting network based on insertion sort or selection sort.

مثال : مثالی برای شرح مرتب سازی به روش ترکیبی درج و انتخاب با ۶ ورودی (n=6).



Cost(6)= (6 x 5)/2= 15

Delay(6)= 2 x 6 – 3 = 9

Cost x Delay = 15 x 9 =135

نتیجه گیری : روش ترکیبی درج و انتخاب نسبت به روش Odd-Even سریعتر است.

۷.۴) شبکه های مرتب سازی Batcher (دسته بندی):

این مرتب سازی برمبنای ادغام (Merg) دو لیست مرتب **m** و **m´** می باشد. در این حالت از تکنیک ادغام Even-Odd یا Odd-Even استفاده می شود.

یک merger_(**m, m´**) مداری است که دو دنباله مرتب شده **m** و **m´** را در یک دنباله به طول **m+m´** مرتب می نماید.

بنابراین :

m: $X_0 \le X_1 \le \ldots\ldots \le X_{m-1}$

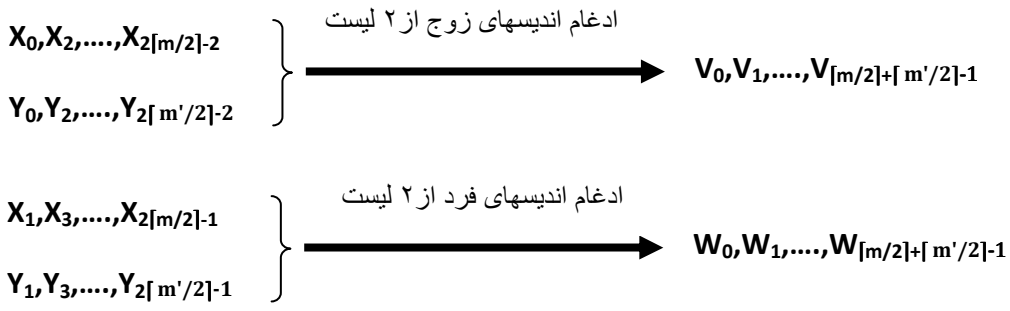
m´: $Y_0 \le Y_1 \le \ldots\ldots \le Y_{m'-1}$

اگر m=0 و **m´=0** باشد عملیاتی انجام نمی گیرد .

اگر **m´=1** =m از یک مقایسه کننده تکی می توانیم استفاده کنیم.

اگر $mm' > 1$ الگوریتم زیر انجام می پذیرد :

الگوریتم ادغام Odd-Even با ادغام عناصر با اندیس زوج- و فرد از دو لیست انجام می گیرد.



در مرحله بعدی زوج عناصر ... **$W_0:V_1, W_1:V_2, W_2:V_3,$** را با هم مقایسه کرده و دنباله **$V_0W_0V_1W_1\ V_2W_2,\ldots$** ایجاد می گردد که کاملا مرتب است .

مثال : دو لیست ۴ و ۷ عنصره زیر را با استفاده از الگوریتم فوق مرتب نمایید.(تکنیک ادغام Odd-Even)

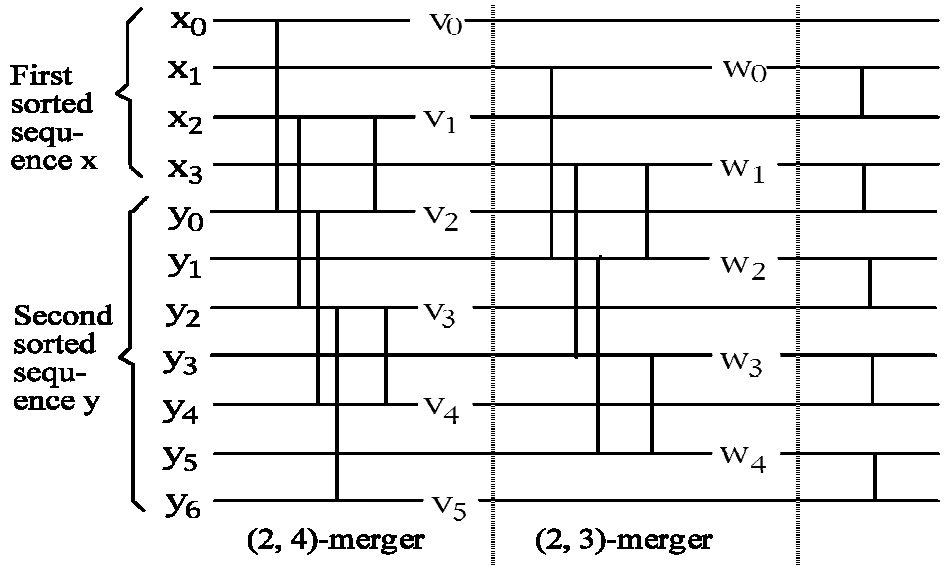


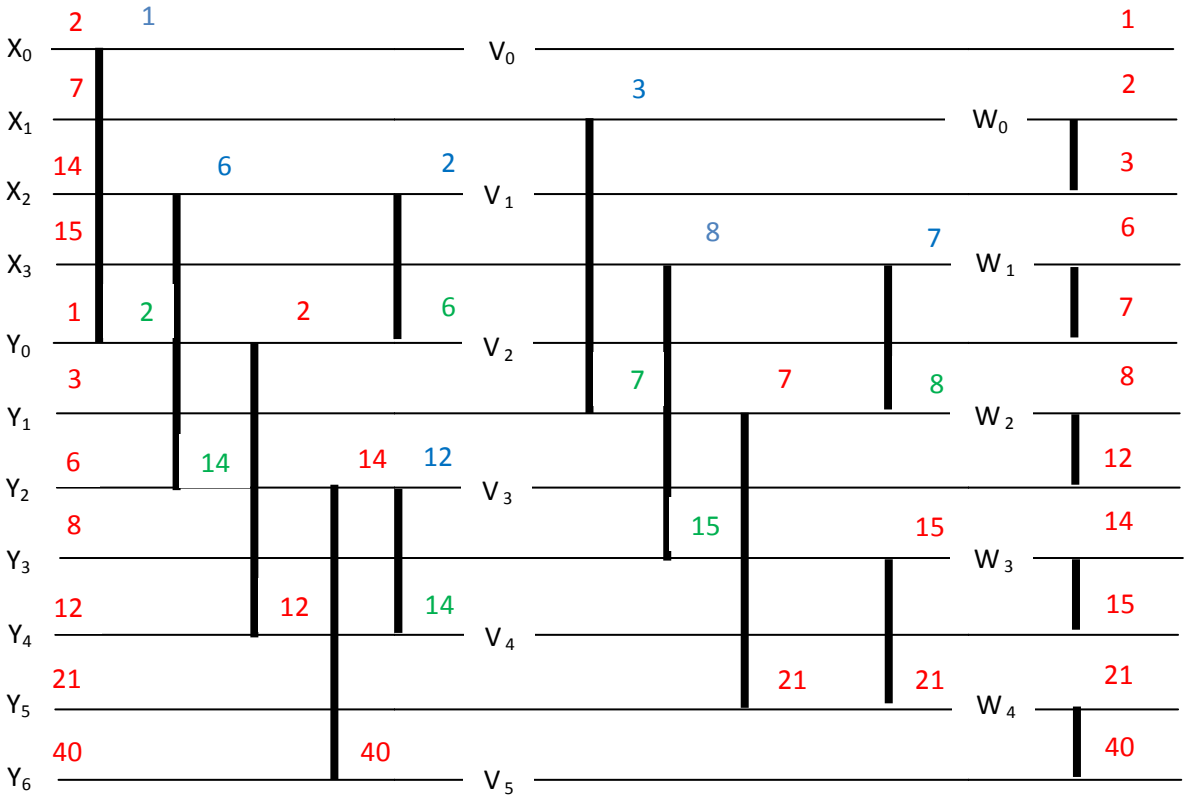
Fig. 7.9 Batcher's even–odd merging network for 4 + 7 inputs.

ابتدا عناصر زوج از هر دو لیست با هم مقایسه می گردند (x_0 با y_0 و x_2 با y_2 و چون عناصر لیست y بیش از عناصر لیست x می باشد y_0 با y_4 و y_2 با y_6 این دو مقایسه آخر سبب خواهد شد که x_0 با y_4 و x_2 با y_6 نیز مقایسه شودو در انتها x_2 با y_0 و y_2 با y_4 مقایسه می گردند) . این مقایسه ها نتایج میانی V_0 تا V_5 را تولید خواهد کرد.

سپس عناصر فرد دو لیست همانند فوق مقایسه خواهند شد و نتایج میانی W_0 تا W_4 تولید می گردد .

و در گام آخر نتایج میانی (W_0 با V_1 و W_1 با V_2 و W_2 با V_3 و W_3 با V_4 و W_4 با V_5) با هم مقایسه می گردند.

مثال عددی :



ساختار بازگشتی (Even_Odd_merger) با شبکه مرتب سازی :

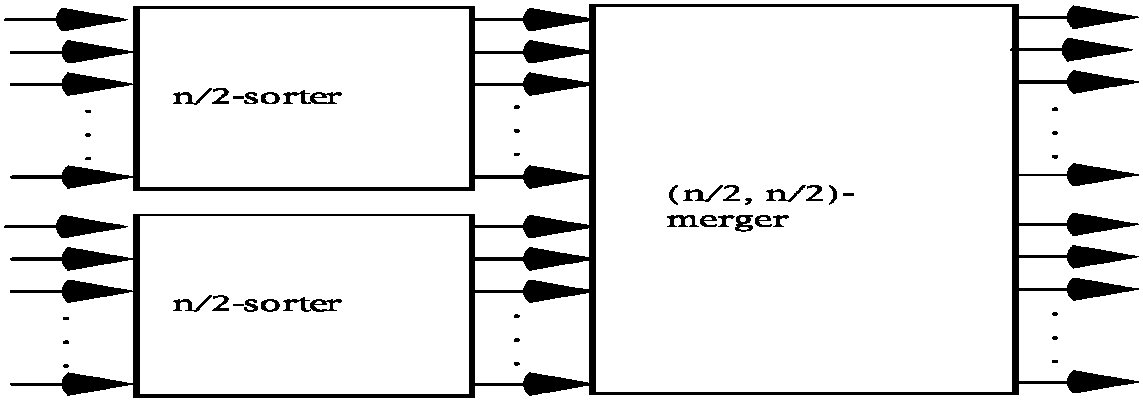


Fig. 7.10 The recursive structure of Batcher's even–odd merge sorting network.

این شبکه ها را از سمت چپ می توان به همین ترتیب نصف کرد تا به یک سری 2_sorter رسید.

مثال : رسم دیاگرام شبکه مرتب سازی بازگشتی Even_Odd_Merger با ۸ ورودی :

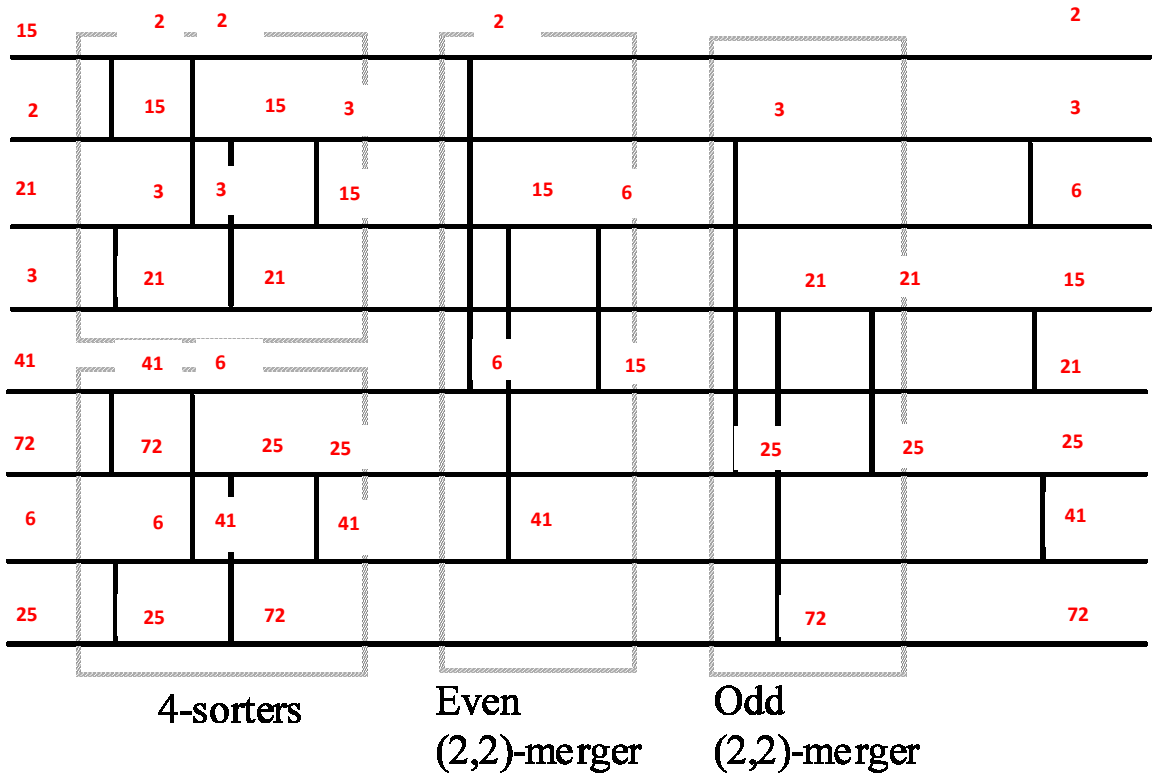


Fig. 7.11 Batcher's even-odd merge sorting network for eight inputs .

همانطور که ملاحظه می گردد ابتدا ۲ عدد 4_sorter (اولی ۴ عنصر ابتدایی و دومی ۴ عنصر انتهایی) را مرتب نموده و سپس Even_merger عناصر زوج ۲ لیست را مرتب و Odd_merger عناصر فرد ۲ لیست را مرتب می نماید و در انتها خروجی های Even و Odd با هم ادغام و کل لیست مرتب می گردد.

۸- مثالهایی از دیگر سطوح چرخشی (Other Circuit-Level Examples) :

بحث مرتب سازی و شبکه های مرتب سازی با موارد زیر کامل می گردد :

- ✓ جستجو – محاسبات Perfix موازی – تبدیلات فوریه .
- ✓ ماشین دیکشنری – شبکه های Perfix موازی .

۸.۱. جستجو و عملیات مربوط به فرهنگ لغات :

الگورینم جستجوی باینری به صورت موازی :

فرض کنید از لیست مرتب X می خواهیم عنصر y را بصورت موازی جستجو کنیم .

۱) این الگوریتم در K مرحله به پایان می رسد که K بصورت زیر محاسبه می گردد(n طول لیست است)

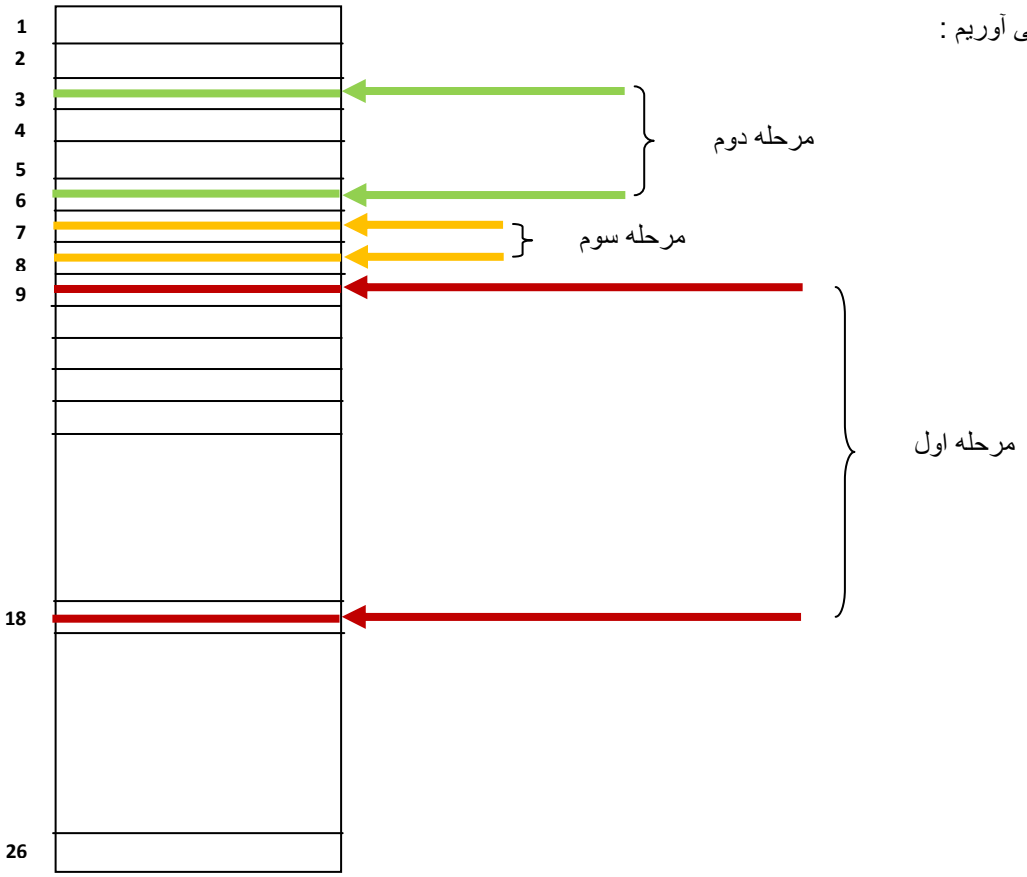
۲) پردازنده **i** ام عنصر **y** (عنصر مورد جستجو) را با $\mathbf{i\ (p+1)^{k-1}}$ امین عنصر مقایسه می کند(p تعداد پردازنده ها و k تعداد مراحل می باشد) . اگر یکی از پردازنده ها **y** را پیدا نمود ، الگوریتم خاتمه می یابد . در غیر اینصورت لیست X به قطعه های $\mathbf{(p+1)^{k-1}-1}$ تقسیم بندی می کند.

۳) مجددا مراحل فوق تکرار می گردند تا یا عنصر y پیدا شده و یا k=1 گردد و این به این مفهوم است که دیگر عنصری برای جستجو وجود ندارد.

تذکر : رابطه $\mathbf{(p+1)^{k-1}}$ در هر مرحله طول حرکت را مشخص میکند .

مثال : فرض کنید که توسط ۲ پردازنده (p=2) میخواهیم عنصر y را در لیست مرتبی حاوی ۲۶ عنصر(n=26) جستجو کنیم :

ابتدا K را بدست می آوریم :



$\left\lceil \frac{\log_2(26+1)}{\log_2(2+1)} \right\rceil = 3$ پس در این مثال در ۳ مرحله عملیات جستجو انجام می گیرد :

مرحله اول :(Step 1) : طول (اندازه)حرکت برابر است با $\mathbf{(2+1)^{3-1} = 9}$:

طول (اندازه)حرکت برابر است با $\mathbf{(2+1)^{3-1} = 9}$: پس پردازنده اول باید از نقطه شروع به اندازه یک برابر طول و پردازنده دوم به اندازه ۲ برابر طول به جلو در لیست حرکت کند . یعنی P_0 عنصر ۹ ام و P_1 عنصر ۱۸ ام از لیست X را با Y مقایسه می کند.

اگر عنصر y در ۲ مقایسه فوق پیدا شد که الگوریتم خاتمه می پذیرد در غیر اینصورت لیست به قطعه های $\mathbf{(p+1)^{k-1} - 1}$ تقسیم می گردد .(فرض کنیم عنصر y در مرحله اول پیدا نشد)

لیست به قطعه های $\mathbf{(2+1)^{3-1}-1=8}$ تقسیم می گردد. (فرض کنیم مقدار y کوچکتر از مقدار عنصر ۹ ام در لیست X می باشد یعنی باید آنرا در ۸ خانه اول لیست جستجو نمود)

حال مرحله فوق را به گونه ای تکرار می کنیم که طول لیست برابر است با (n=8)

مرحله دوم :(Step 2)

K جدید را بر اساس طول لیست جدید بدست می آوریم : $\left\lceil \frac{\log_2(8+1)}{\log_2(2+1)} \right\rceil = 2$ یعنی K=2 پس خواهیم داشت :

طول (اندازه)حرکت برابر است با $\mathbf{(2+1)^{2-1} = 3}$: پس پردازنده اول باید از نقطه شروع (در این مرحله Offset=9)به اندازه یک برابر طول و پردازنده دوم به اندازه ۲ برابر طول به عقب از محل Offset در لیست حرکت کند (بنا به فرض محله قبل). یعنی P_0 عنصر (9-6=3) ام و P_1 عنصر (9-3=6) ام از لیست X را با Y مقایسه می کند.

اگر عنصر y در ۲ مقایسه فوق پیدا شد که الگوریتم خاتمه می پذیرد در غیر اینصورت لیست به قطعه های $\mathbf{(p+1)^{k-1} - 1}$ تقسیم می گردد .(فرض کنیم عنصر y در مرحله دوم نیز پیدا نشد و مقدار y بزرگتر از مقدار عنصر ۶ ام در لیست X می باشد)

لیست به قطعه های $\mathbf{(2+1)^{2-1}-1=2}$ تقسیم می گردد.

حال مرحله فوق را به گونه ای تکرار می کنیم که طول لیست برابر است با (n=2)

مرحله سوم :(Step 3)

K جدید را بر اساس طول لیست جدید بدست می آوریم : $\left\lceil \frac{\log_2(2+1)}{\log_2(2+1)} \right\rceil = 1$ یعنی K=1 پس خواهیم داشت :

طول (اندازه)حرکت برابر است با $\mathbf{(2+1)^{1-1} = 1}$: پس پردازنده اول باید از نقطه شروع (در این مرحله Offset=6)به اندازه یک برابر طول و پردازنده دوم به اندازه ۲ برابر طول به جلو از محل Offset در لیست حرکت کند (بنا به فرض محله قبل). یعنی P_0 عنصر (6+1=7) ام و P_1 عنصر (6+2=8) ام از لیست X را با Y مقایسه می کند.

اگر عنصر y در ۲ مقایسه فوق پیدا شد که الگوریتم خاتمه می پذیرد در غیر اینصورت عنصر y در لیست X وجود ندارد.

عملیات دیکشنری :

برای اینکه کاربردهای عملی جستجو را بیان کنیم مجموعه ای از عملیات قابل انجام روی دیکشنری را تعریف می کنیم با این فرض که رکوردی به طول n با کلیدهای $X_0,X_1,...,X_{n-1}$ داریم :

- ✓ پیدا نمودن رکورد با کلید Y و برگرداندن رکورد مر بوطه : **Search(Y)**
- ✓ درج رکود Z با کلید Y در داخل یک لیست : **Insert(Y,Z)**
- ✓ حذف رکورد با کلید Y(در برخی از مواقع داده حذف شده نیز برگردانده می شود): **Delete(Y)**
- ✓ پیدا کردن رکورد با کوچکترین کلید: **FindMinimom**
- ✓ پیدا کردن رکورد با بزرگترین کلید: **FindMaximom**
- ✓ پیدا کردن رکورد با کلید میانه (کلید وسطی): **FindMinimom**
- ✓ پیدا کردن رکورد با نزدیکترین کلید به کلید Y: **Findbest(Y)**
- ✓ پیدا کردن رکورد بعد از کلید Y: **Findnext(y)**
- ✓ پیدا کردن رکورد قبل از کلید Y: **Findprev(y)**
- ✓ حذف رکورد حداقل از لیست : **Extractmin**
- ✓ حذف رکورد حداکثر از لیست : **Extractmax**
- ✓ حذف رکورد میانه از لیست : **Extractmed**

ماشین دیکشنری با ساختار درختی :

این ماشین به منظور انجام عملیات فوق به صورت موازی ساخته شده است .

ماشین از ۲ درخت باینری کامل (back-to-back) که برگهای آنها با هم ادغام شده تشکیل شده است .(درخت دایره و درخت مثلث در هم ادغام می شوند)

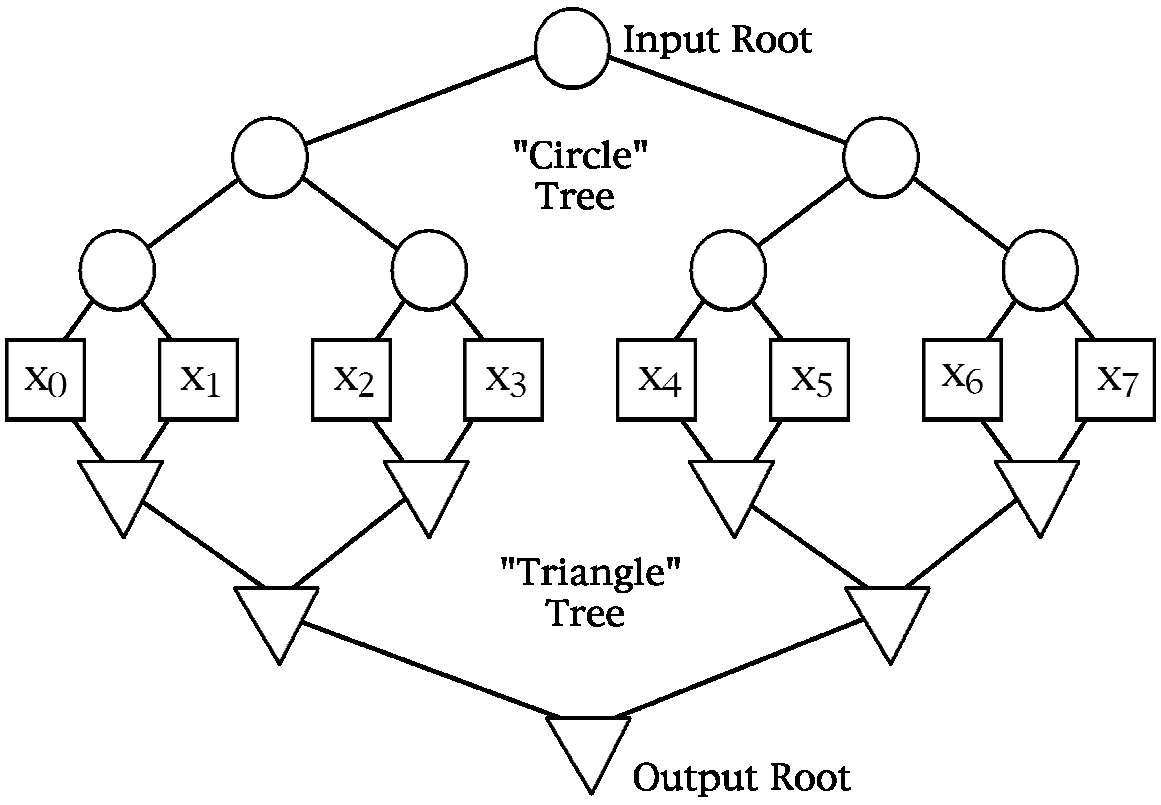


Fig. 8.1 A tree-structured dictionary machine.

۱) درخت "دایره" برای انتشار عملیات دیکشنری که توسط ریشه (Input Root) وارد می شود به تمامی گره های برگها که رکوردها را در خود ذخیره می کنند استفاده می گردد.(مثلا اگر عملیات search یک رکورد باشد با صفر و یک مشخص می کند. اگردر برگ پیدا کرد ۱ و اگر پیدا نکرد صفر در نظر می گیرد)

۲) درخت "مثلث" نتایج عملیات مختلف را از گره های برگ ترکیب نموده و نتیجه نهایی را به Output Root انتقال می دهد.(مثلا اگر Search رکورد باشد با OR کردن صفر و یکهای برگها نتیجه را به Output Root انتقال می دهد)

مثال : فرض کنید می خواهیم کلید Y را در یک دیکشنری با ۸ کلید جستجو کنیم . نحوه پیمایش درخت به قرار زیر است :

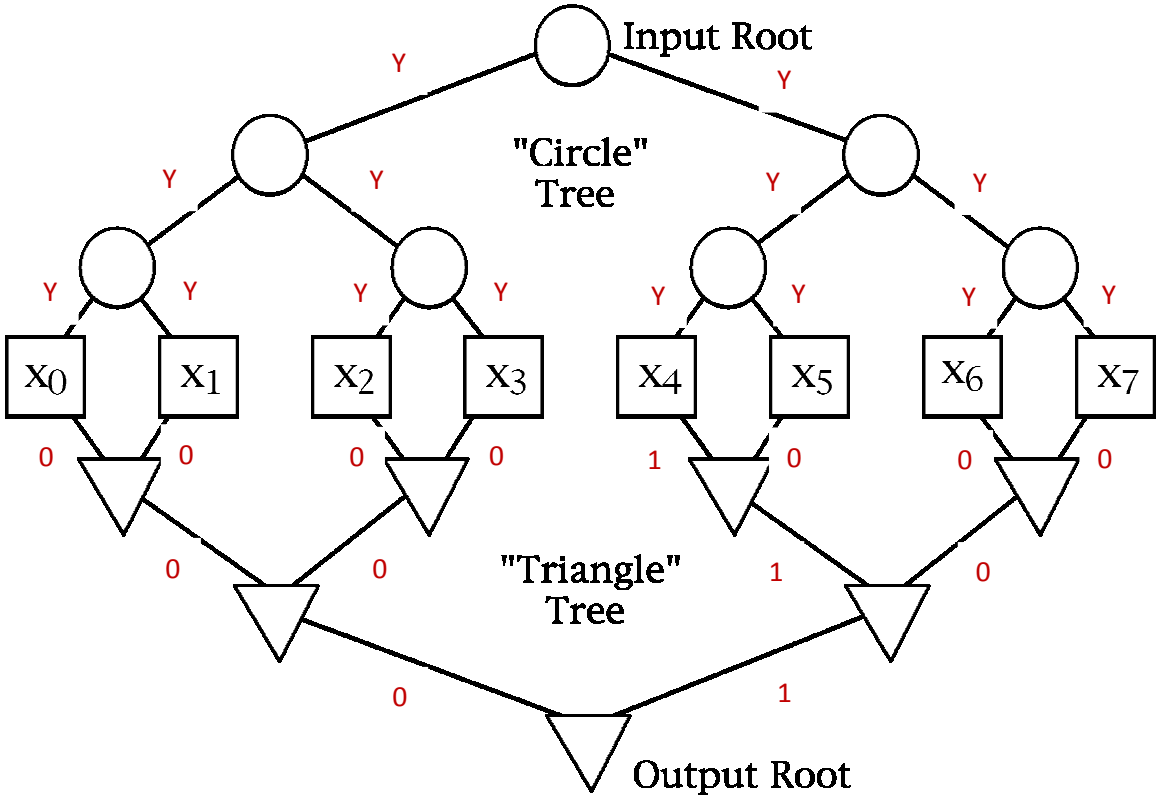


Fig. 8.1 A tree-structured dictionary machine.

ابتدا کلید Y در کلیه گره های درخت "دایره" پخش می گردد تا به برگها برسد .

در صورتی که مقدار کلید Y با برگ X یکسان باشد مقدار 1 و در غیر اینصورت مقدار 0 به گره های درخت "مثلث" منتقل می گردد(فرض کنیم کلید Y با برگ X_4 یکسان است)

گره های درخت "مثلث" با OR کردن مقدیر صفر و یک نتیجه را به گره های پایین خود ارسال می کنند تا نتیجه در Output Root نمایان گردد(اگر 1 باشد به این معنی است که کلید در دیکشنری موجود است)

تذکر: کلید Y می تواند به منظور عملیات دیگر اطلاعات دیگری به همراه خود داشته و همچنین از گرههای درخت "مثلثی" اطلاعات دیگری نظیر دیگر اطلاعات رکورد به Output Root منتقل گردد.

۸.۳. محاسبات موازی Prefix :

در این محاسبات عملگر ⊗ تعریف شده که می تواند جمع یا تفریق یا ... باشد.

پیاده سازی محاسبات موازی Prefix به صورت بازگشتی می باشد.

یک دنباله داده n تایی با مقادیر $X_0,X_1,\ldots X_n$ داریم که $S_i=S_{i-1} \otimes X_i$ (S_0 ورودی اولیه می باشد)

مثال : Prefix_Sum ($S_0=0+X_0$)

Example: Prefix sums

x_0	x_1	x_2	. . .	x_i
x_0	$x_0 + x_1$	$x_0 + x_1 + x_2$	. . .	$x_0 + x_1 + \ldots + x_i$
s_0	s_1	s_2	. . .	s_i

Sequential time with one processor is O(*n*)
Simple pipelining does not help

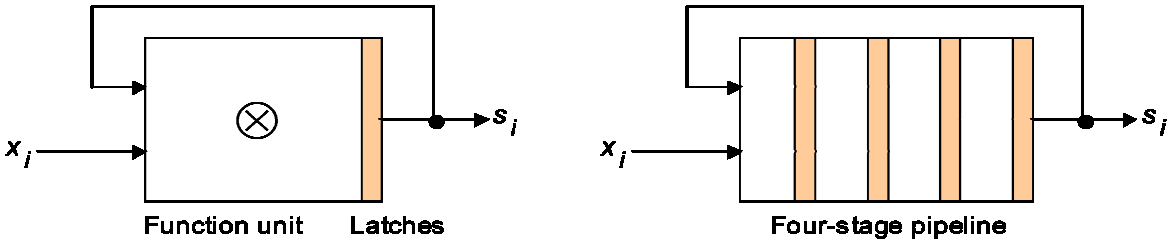


Fig. 8.4 Prefix computation using a latched or pipelined function unit.

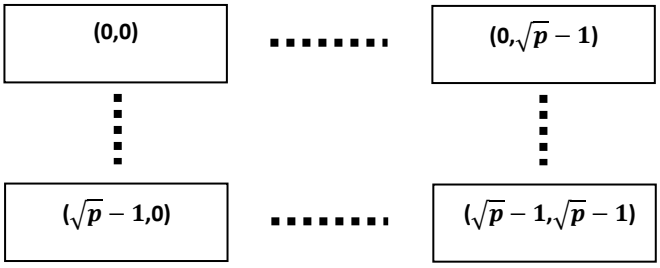
قسمت سوم (معماریهای بر مبنای مش)

فصل نهم از قسمت سوم : مرتب سازی با شبکه های مش ۲ بعدی (2D Mesh)

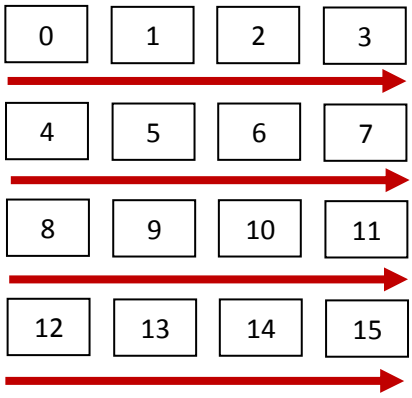
۹.۱. کامپیوترهای متصل شده بصورت مش (انواع شماره گذاری پردازنده های مش ۲ بعدی)

✓ **استفاده از شماره های سطر و ستون برای نام گذاری پردازنده ها :**

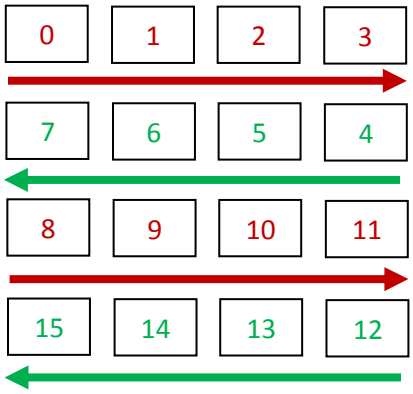
پردازنده ها در مش ۲ بعدی به روشهای مختلف شماره گذاری می شوند.ساده ترین و طبیعی ترین را ه برای نام گذاری پردازنده ها استفاده از شماره سطرها و ستونها می باشد.بنابراین پردازنده با شماره **(0,0)** اشاره به گوشه بالا در سمت چپ مش و پردازنده با شماره $(0,\sqrt{p}-1)$ اشاره به گوشه سمت راست بالای مش و پردازنده با شماره $(\sqrt{p}-1,\sqrt{p}-1)$ اشاره به گوشه سمت راست پایین مش می نماید(P تعداد پردازنده های مش می باشد).



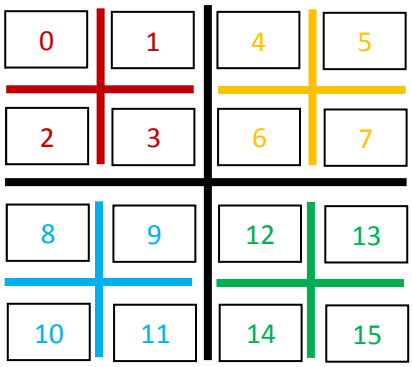
✓ **شماره گذاری خطی (Row Major):** در این روش پردازنده ها بصورت خطی و از چپ به راست شماره گذاری می گردند.



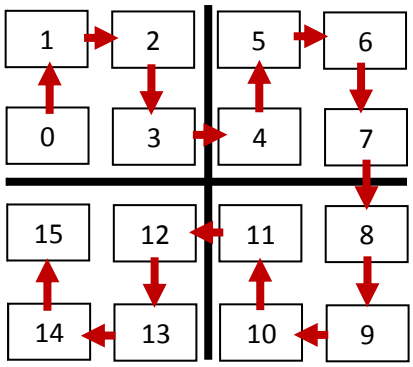
✓ **شماره گذاری خطی شبیه مار (Snake Like Row Major):** در این روش پردازنده ها بصورت خطی پیچشی (مثل حرکت مار) شماره گذاری می شوند.



✓ **شماره گذاری خطی کنارهم گذاشتن (Shuffled Row Major):** در این روش پردازنده ها ابتدا بصورت چهارگوش تقسیم شده و بعد بصورت خطی شماره گذاری می گردند.



✓ **شماره گذاری براساس مجاورت (Proximity Order):**



نحوه ارتباط بین پردازنده های مش ۲ بعدی :

ارتباط بین پردازنده ها در یک مش ۲ بعدی از طریق ثباتها (رجیستر) موجود در این پردازنده ها صورت می پذیرد.

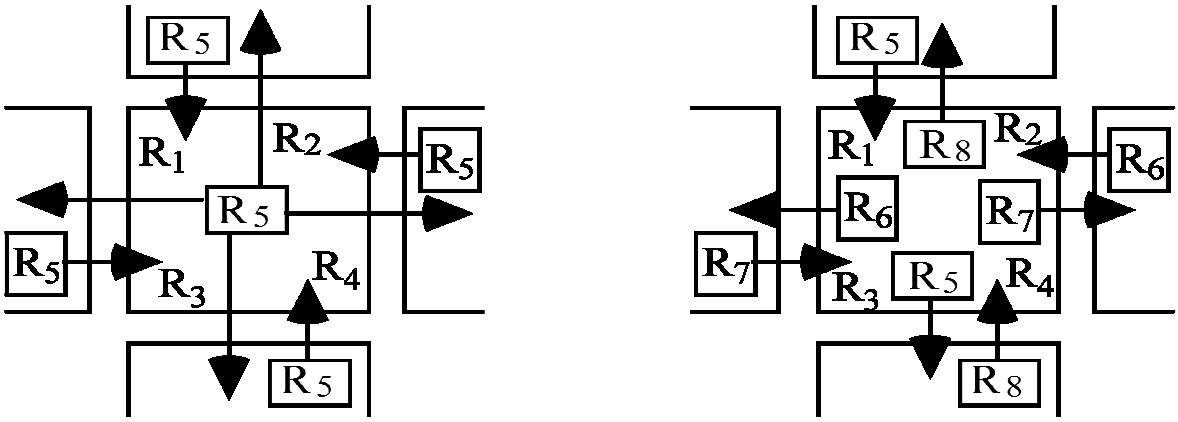


Fig. 9.4 Reading data from NEWS neighbors via virtual local registers.

در شکل سمت چپ ثبات R_5 از پروسسور ارسال کننده داده خود را در اختیار پردازنده های مجاورخود قرار می دهد و از طریق ثباتهای R_1 تا R_4 داده ها را از پردازنده های مجاور در یافت می نماید.(یک ثبات برای ارسال و ۴ ثبات برای دریافت)

ولی در شکل سمت راست ۴ ثبات برای ارسال و ۴ ثبات نیز برای دریافت اطلاعات از پردازنده های مجاور وجود دارد.

۹.۲. الگوریتم مرتب سازی به روش Shearsort :

در این الگوریتم در صوتی که یک مش با ابعاد r x (p/r) (r تعداد سطرها و p تعداد پردازنده ها) داشته باشیم برای مرتب سازی لازم است $K=\lceil \log_2 r \rceil + 1$ مرحله را طی نماییم.

مراحل الگوریتم :

مرحله اول : ابتدا مش را بصورت مارپیچ (Snake-like) مرتب می کنیم .یعنی از روی مش بصورت مارپیچ عبور کرده و به هنگام عبور عناصر هر سطر را مرتب می کنیم.(اگر ردیف سطرها از صفر شروع شود سطرهای زوج از چپ به راست و سطرهای فرد از راست به چپ مرتب می شوند)

مرحله دوم : سپس کلیه ستونهای مش را از بالا به پایین مرتب می کنیم.

تذکر : مراحل اول و دوم به تعداد $\lceil \log_2 r \rceil$ مرتبه تکرار می شود.

مرحله آخر : بر اساس اینکه شماره گذاری پردازنده ها در مش بصورت مارپیچ (Snake-like) یا سطری (Row Major) انجام شده باشد یکی از روشهای مارپیچ یا سطری برای مرتب سازی انجام می شود.

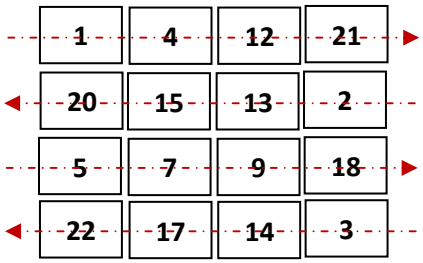
مثال : یک مش مربعی در ابعاد (4x4) با داده های زیر را در نظر گرفته و آنرا به روش Shearsort مرتب نمایید.

1	12	21	4
15	20	13	2
5	9	18	7
22	3	14	17

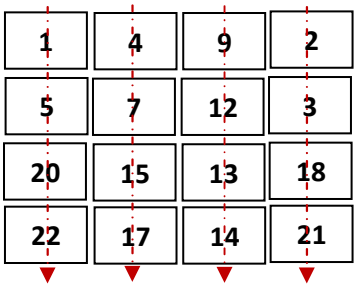
حل : تعداد مراحل تکراری (مراحل اول و دوم) برابر است با : $K= \lceil \log_2 4 \rceil =2$

چرخش اول K=1 :

مرحله اول : سطرها به روش مارپیچ مرتب می شوند :

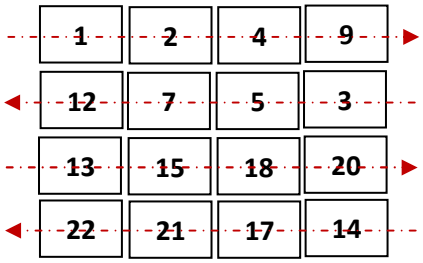


مرحله دوم : ستونها به روش بالا به پایین مرتب می شوند :

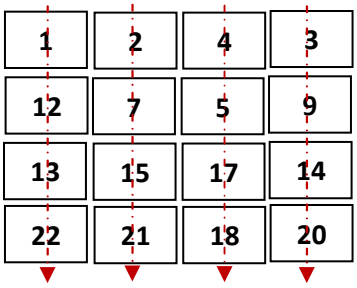


چرخش دوم K=2 :

مرحله اول : سطرها به روش مارپیچ مرتب می شوند :

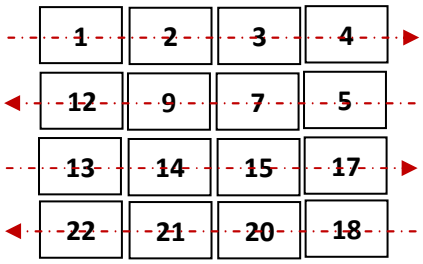


مرحله دوم : ستونها به روش بالا به پایین مرتب می شوند :

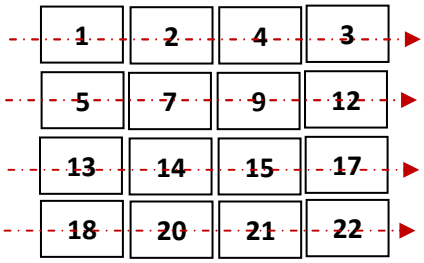


مرحله آخر K=2+1=3 :

اگر چینش پردازنده ها بصورت مارپیچ باشد:



اگر چینش پردازنده ها بصورت سطری (Row major) باشد



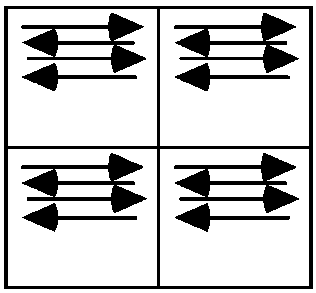
۹.۳. الگوریتم مرتب سازی بازگشتی :

الگوریتم مرتب سازی بازگشتی روی یک مش مربع در ابعاد $\sqrt{p}\ X\ \sqrt{p}$ که به ۴ قسمت تقسیم می شود براساس استراتژی تقسیم و غلبه کار می کند.

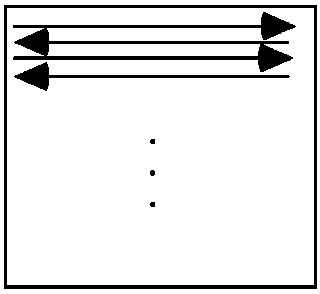
این الگوریتم از ۴ مرحله زیر پیروی می کند:

۱) مرتب سازی هر یک از ۴ ربع بر اساس مرتب سازی مارپیچ (Snake-like)
۲) مرتب سازی سطرها به طور مستقل به صورت مارپیچ (Snake-like)
۳) مرتب سازی ستونها بطور مستقل از بالا به پایین
۴) اعمال $4\sqrt{p}$ مرحله از odd–even transposition روی داده های سرتاسری بصورت مارپیچ .

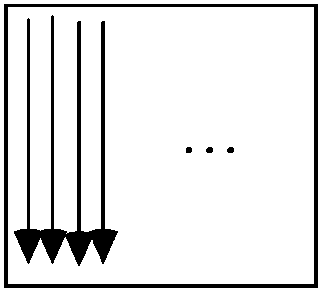
9.4 Recursive Sorting Algorithms



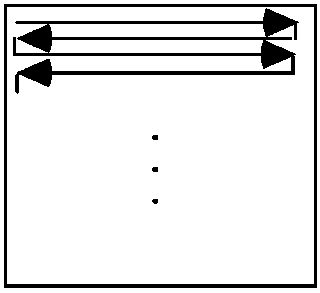
1. Sort quadrants



2. Sort rows



3. Sort columns

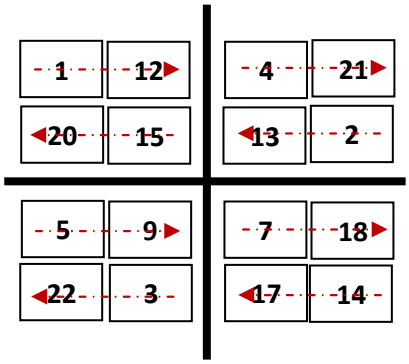


4. Apply $4\sqrt{p}$ steps of odd-even transposition along the overall snake

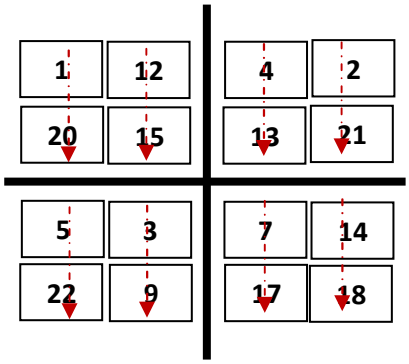
مثال : مرتب سازی مش زیر به روش بازگشتی :

1	12	21	4
15	20	13	2
5	9	18	7
22	3	14	17

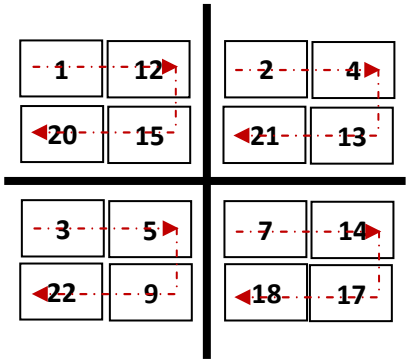
مرحله ۲و۱ : تقسیم مش فوق به ۴ قسمت و مرتب سازی مارپیچ هر قسمت :



مرحله ۳ : مرتب سازی ستونها بطور مستقل از بالا به پایین :



مرحله ۳ : اعمال $4\sqrt{p} = 16$ مرحله از odd–even transposition روی داده های سرتاسری بصورت مارپیچ :



پایان