فصل ۸

مثالهایی از دیگر سطوح چرخشی

در این فصل ما به مطالعه درس هزینه کاربردی به همراه معماری های موازی که برای آدرس دهی نیازهای خاص توسعه داده شد می پردازیم. زمینه های برنامه کاربردی شامل عملیاتهای لغتنامه ،محاسبه پیشوند موازی، و تبدیل فوریه سریع می باشد. در نتیجه ساختارهای سخت افزاری (دستگاه درخت ، شبکه های پیشوندی موازی ، و شبکه پروانه) به طور بسیار بالایی برای برنامه های کاربردی در دست اختصاص داده شده و ممکن است در برخورد با مسائل دیگر مناسب نباشد. بخشهای مشترک از طریق این فصل عبارت است از مرتب سازی شبکه ها در فصل ۷ ، معماری هایی که به طور کامل در سطح مداری مشخص شده اند ؛ به طور مثال ، ساختار داخلی هر پردازنده قابل کشیدن از مدار دیجیتالی نمونه می باشد، ارئه سادگی در مورد توابع کنترلی و دستکاری های داده. به طور مشابه ، اتصالات داخلی پردازنده به طور کامل شرح داده شده و به راحتی قابل تحقق با سربار و تاخیر محیط متوسط می باشد. موضوعات فصل شامل :

- ۸,۱ جستجو و عملیاتهای لغتنامه ای
- ۸,۲ ماشین لغتنامه ساختار درختی
- ۸,۳ محاسبه پیشوندی موازی
- ۸,۴ شبکه های پیشوندی موازی
- ۸,۵ تبدیل فوریه گسسته
- ۸,۶ معماری های موازی برای FFT

۸,۱ جستجو و عملیاتهای لغتنامه ای

جستجو کردن یکی از مهمترین عملیاتها در کامپیوتر های دیجیتال بوده و بخش بزرگی از منابع را مصرف می کند. یک دلیل اصلی برای مرتب سازی ، فعالیتی که تخمین زده شده است که بیش از یک چهارم زمان اجرا را در بیشتر کامپیوتر ها مصرف می کند. بنابراین ، این اغراق نیست که بگوییم جستجو کردن حداقل ۲۵ درصد از زمان اجرا را در بیشتر کامپیوتر ها به صورت مستقیم یا غیر مستقیم مصرف می نماید. بدیهی است سرعت بخشیدن به این عملیات از طریق برنامه کاربردی موازی سازی مطلوب خواهد بود.

اجازه دهید ابتدا ببینیم چطور Y را در لیستی از n تا کلید که قابل شتاب دهی بر روی پردازنده P ، PRAM می باشد را جستجو کنیم. فرض کنید که به عنوان لیست ورودی به صورت صعودی مرتب شده است. به یاد داشته باشید که یک تک پردازنده ای که از الگوریتم جستجوی دودویی برای جستجوی یک لیست مرتب شده که به طور موثر $\lceil \log_2(n+1) \rceil$ مرحله مقایسه دارد استفاده می کند. کلید Y داده شده با کلید $x_{\lfloor n/2 \rfloor}$ در میان یا نزدیک به میان در لیست مقایسه می شود. در صورتی که$y = x_{\lfloor n/2 \rfloor}$ باشد ، آنگاه جستجو تمام است. دو رخداد دیگر محتمل است :

محدود کردن جستجو به X_0 از طریق $x_{\lfloor n/2 \rfloor - 1}$ y<$x_{\lfloor n/2 \rfloor}$

محدود کردن جستجو به $x_{\lfloor n/2 \rfloor + 1}$ از طریقx_{n-1} y>$x_{\lfloor n/2 \rfloor}$

در هر دو حالت بالا ، اندازه مساله به $n/2$ کاهش می باشد که منجر به تعداد مراحل لگاریتمی می شود. توسعه الگوریتم جستجوی دودویی به P+1 جستجو بر روی P پردازنده PRAM سر راست بوده و منجر به زمان اجرای K =

$\lceil \log_2(n+1)/\log_2(p+1) \rceil$مرحله می شود. اثبات از طریق القاء می باشد. $n = (p+1)^k - 1$را برای بعضی از مقادیر K قرار دهید در غیر اینصورت کمترین مقدار K از جمله $n \leq (p+1)^k - 1$ را پیدا نمایید. زمان اجرای ادعا شده می تواند به وضوح برای k=1 (و p تا پردازنده در یک مرحله یک لیست p المانی را جستجو کند) بدست آورد. فرض کنید که k-1 مرحله برای هر لیست از اندازه که بیشتراز $(p+1)^{k-1} - 1$ نیست کافی باشد. در مرحله ۱ ، پردازنده i ، y را با $i(p+1)^{k-1}$ مین عنصر در x مقایسه می کند. در هر صورت یکی از پردازنده های y را بیابد یا جستجو به یک بخش از لیست که اندازه آن بزرگتر از $(p+1)^{k-1} - 1$ نباشد محدود می شود.از این رو ، به طور کلی k مرحله مورد نیاز می باشد.

الگوریتم بالا در صورتی که هیچ الگوریتم جستجوی مبتنی بر مقایسه نتواند سریعتر باشد مطلوب می باشد. ما از القا بر k برای نشان دادن آنکه بعد از k مقایسه موازی ، باید یک قطعه بررسی نشده پیوسته یا بیشتر از لیست x که شامل حداقل $(n+1)/(p+1)^k - 1$کلید است استفاده می کنیم. معادل قرار دادن این عبارت با ۰ ، بازه پایین تر از تعداد k مرحله را نتیجه می دهد. سرعت بالا بدست آمده توسط الگوریتم جستجوی موازی بالا برابر

$$\lceil \log_2(n+1) \rceil / \lceil \log_2(n+1)/\log_2(p+1) \rceil = \log_2(p+1)$$

می باشد. به طوری که کاملا با توجه به بهبود الگوریتم مایوس می باشد.

حتی اگر جستجوهای ساده در یک لیست مرتب شده از کلیدها نتواند به طور قابل توجهی توسط پردازشهای موازی شتاب یابد ، همه امید ها از دست داده نمی شود و جستجوی موازی همچنان مهم می باشد . اولا، در برنامه های کاربردی جایی که لیستی از کلید ها به طور مکرر تغییر می کنند ، نیاز لیست مرتب شده غیر واقعی است و منجر به سربار قابل توجهی که باید در آنالیز کلی برنامه مورد محاسبه قرار گیرد. حافظه ها و پردازنده های مزتبط می تواند به طور کامل کارآمد در جستجوی لیستهای مرتب نشده بوده و به طور معمول شتاب در محتوا را به صورت خطی را منجر می شود (شکل ۴,۱ را ببینید). ثانیا ، و این تمرکز اصلی ما در اینجاست ، زمانی که m تا جستجوی مختلف در لیستی از n تا کلید همسان اجرا می شود ، جستجوی موازی ممکن است کاملا کارآمد بوده و می تواند منجر به شتاب بسیار بهتری شود. این مسئله که توسط یک واژه پرداز چک کننده املاء بررسی شده است تا به هدف تطبیق آنکه لیستی از کلمات سندی در یک فرهنگ لغت استاندارد ظاهر می شود ، که به عنوان جستجوی دسته ای شناخته شده می باشد.

برای قراردادن نیاز برای جستجو در زمینه برنامه های واقع بینانه ، مجموعه ای از عملیاتهای لغتنامه را بروی لیستی از n رکورد داده شده با کلیدهای $x_0, x_1, \ldots, x_{n-1}$ تعریف می کنیم. ما می خواهیم به طور کارآمد سهعملیات پایه ای را برای اجرا فعال کنیم. در بیشتر موارد به شرح زیر "the record" می تواند با "a record" یا "all records" در صورتی که رکوردهای چندگانه با یک مقدار کلید مساوی می تواند در میان رکورده های ذخیره شده موجود جایگزین گردد.

Search(y) یافتن رکورد با کلید y و برگرداندن داده مرتبط آن

Insert(y,z) تکمیل لیست با رکوردی که شامل کلید y و بخش داده z است

Delete(y) حذف رکورد با کلید y [و ، به طور اختیاری ، برگرداندن داده مرتبط]

عملیات delete(y) در صورتی که هیچ رکوردی با مقدار کلید y یافت نشود ، زائد گفته می شود. درچنین موردی ،عملیات میتواند به طور ساده ای چشم پوشی شده و یا ممکن است یک استثناء را اعلام نماید.به طور مشابه ، عملیات Insert(y,z) در صورتی که رکوردی با مقدار کلید y در حال حاضر ذخیره شده و لیست رکوردها محدود به نگهداری فقط یک رکورد با مقدار کلید داده شده باشد ، زائد گفته می شود. در این مورد ، ما اغلب "insert" را به معنای "به روز رسانی (update) یا تعویض بخش داده رکورد با مقدار کلید y تا z" اگرچه چشم پوشی وارد کردن زوائد همچنان یک گزینه می باشد به کار می بریم.

علاوه بر این ، همه یا بعضی از عملیاتهای زیر ممکن است مورد علاقه باشد:

Findmin یافتن رکورد با کوچکترین ارزش کلیدی و بازگشت داده مرتبط با آن

Findmax یافتن رکورد با بزرگترین ارزش کلیدی و بازگشت داده مرتبط با آن

findmed یافتن رکورد با مقدار متوسط کلیدی و بازگشت داده مرتبط با آن

Findbest(Y) پیدا کردن رکورد با در نظر گرفتن بهترین و یا نزدیکترین به ارزش کلیدیY

Findnext(Y) پیدا کردن رکوردی که کلید آن جانشین ارزش کلیدی Y می باشد

Findprev(Y) پیدا کردن رکوردی که کلید آن اسبق ارزش کلیدی Y می باشد

Extractmin حذف رکورد (ها) با کوچکترین مقدار کلیدی [بازگرداندن داده رکورد]

Extractmax حذف رکورد (ها) با بزرگترین مقدار کلیدی [بازگرداندن داده رکورد]

Extractmed حذف رکورد (ها) با مقدار متوسط کلیدی [بازگرداندن داده رکورد]

عملیاتهای findmin و extractmin (یا findmax و extractmax) بعضی اوقات به عنوان عملیاتهای صف اولویت معرفی می شوند. برای مثال ، کمترین مقدار کلید ممکن است مربوط به وظیفه ای با بالاترین اولویت در صف وظایف که آماده برای اجرا می باشند باشد. یافتن سطح اولویت وظیفه با بالاترین اولویت در صف ممکن است زمانی که می خواهیم تصمیم بگیریم که آیا وظیفه در حال اجرای جاری می تواند ادامه یابد یا باید پیشدستی شود مورد علاقه باشد. استخراج وظیفه با بالاترین اولویت از صف اولویت زمانی که می خواهیم وظیفه ای را برای اجرا بر روی پردازنده بیکار زمانبندی نماییم مورد نیاز می باشد.

۸,۲ یک لغتنامه ساختار درختی

در برخورد با طرحهای سطح –مدار ، ما اساسا در جهتی که معکوس آنچه که در بقیه کتاب انجام شده است را : به جای تعریف یک " هدف کلی " معماری موازی (به طور مثال PRAM و 2D mesh و Hypercube) و توسعه الگوریتم برای حل مسائل متنوع مورد علاقه بر روی آن ، ما مسئله ای را انتخاب و یک یا بیشتر تحقق مداری یا معماری که می تواند آن را در حالت موازی به طور کارآمد حل نماید برخورد می نماید را ادامه می دهیم.

ماشینهای لغتنامه ساختار درختی ، نمونه ای است که در این بخش شرح داده شده، که توسط Bentley و kung پیشنهاد شد، در میان دیگران ، و بعد از آن توسط محققین دیگر توسعه و تصحیح گردید. برخی دیگر پیشنهاد ماشینهای لغتنامه مبتنی بر مش ها ، hypercubes و غیره (نگاه کنید بهمثالهای [parh90] و [Nara96] برای مراجع و آخرین تحولات در ماشینهای لغتنامه) دادند.

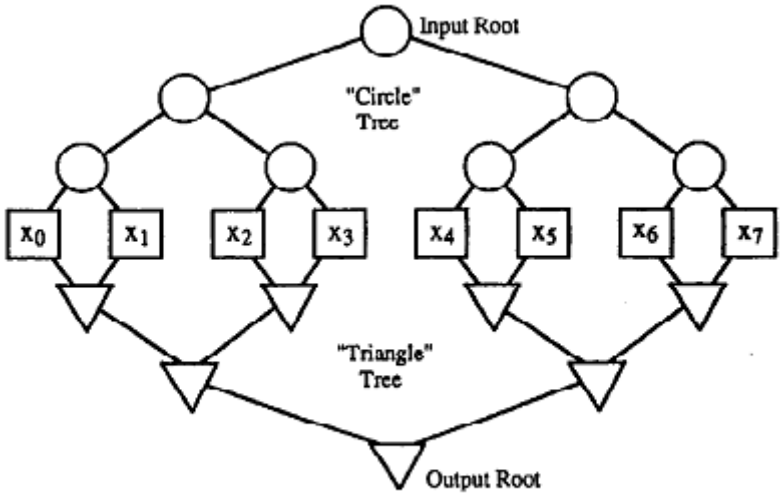

Figure 8.1. A tree-structured dictionary machine.

ماشین درخت شامل دو درخت باینری کامل پشت به پشت که برگهایش ادغام شدهاند (شکل ۸,۱) می باشد.در خت "Circle" مسئول پخش عملیاتهای لغتنامه که از طریق "ریشه ورودی" به تمام گره های برگ که رکوردها را نگه می دارد می باشد. درخت "Triangle" نتایج عملیاتهای فردی توسط گره های برگ را به داخل نتیجه کلی که منجر به "ریشه خروجی" می شود ترکیب می کند.در یک پیاده سازی سخت افزاری وافعی ، درخت دوبل ممکن است با گره های دایره ای و مثلثی ادغام شده مانند گره های برگ fold شده باشد.با این حال ، حتی با fold کردن ، دو تابع جداگانه از پخش کردن و ترکیب کردن انجام خواهد شد ، ساخت ارائه شکل ۸,۱ بسیار مناسبتر از یک درخت باینری ساده می باشد.

جستجو می تواند به راحتی از طریق سطوح دو درخت در شکل ۸,۱ لوله ای شوند. به عنوان اولین دستور ، $Search(y_0)$ ، از ریشه ورودی به دو فرزندش می گذرد ، و مورد بعدی $search(y_1)$ می تواند قابل قبول باشد. دو دستور العمل در Lockstep لوله ای شد ، در نهایت رسیدن به گره های برگ در چرخه های پی در پی می شود. هر پردازنده برگ مسئول دستور العمل جستجو ، با پاسخ جستجو (به عنوان مثال ، yes/no ، ID گره برگ ، مقدار کلید ، داده یا یک شاخص تطبیق درجه در مورد بهترین تطبیق جستجو) ترکیب شده ، در همان مد لوله شده در درخت پایین تر می باشد.

تابع ترکیب گره های مثلثی بستگی به نوع جستجو دارد. در عملیاتهای زیر ممکن است ID گره برگ (که نشان دهنده محل گره برگ جایی که تطابق پیدا شده) همراه با کنترل اطلاعات و دادع ، به عنوان بخشی از نتیجه جستجو نوشته می شود.

Search(y) ارسال OR منطقی سیگنال yes همراه با داده از طرف yes یا یکی از دو طرف در صورتی که هر دو طرف yes را نشان دهند (انتخاب می تواند چرخشی یا اولویتی باشد)

Findmin ارسال کوچکترین مقدار در بین دو کلید به همراه داده مرتبط

Findmax شبیه به findmin

Findmed عملیات یافتن میانهتوسط این طراحی کلی پشتیبانی نمی شود

Findbest(y) ارسال بزرگترین بین دو شاخص منطبق الدرجه به همراه رکورد

Findnext(y) گره های برگ بیت های بزرگتری را تولید می کنند ؛ findmin بین مقادیر بزرگتر انجام می شود.

Findprev(y) شبیه به findnext

در صورتی که nتا گره برگ موجود باشد ، دستور العمل search در تعداد مرحله $[\log_2 n]$بعد از آنکه وارد گره ریشه ورودی شده و نتیجه اش از گره ریشه خروجی پس از $[\log_2 n]$بار مرحله بیشتر ظهور یافت به برگها می رسد. در نتیجه مجموعه زمان تاخیر $2[\log_2 n] + 1$ مرحله با ظهور یک نتیجه جستجو در هر پرخه بعدی می شود. به این ترتیب ، توان عملیاتی می تواند یک جستجو در هر چرخه را با رویکرد مناسب جستجوهای دسته ای بزرگ انجام دهد . در صورتی که جستجو ها تنها عملیات مورد علاقه و لیست رکوردها استاتیک باشد، سرعت بالا نسبت به جستجو دودویی متوالی قابل ملاحظه نخواهد بود. ما در پی استفاده از 3n-2 گره یا پردازنده های ساده (دایره ای ، مربعی و مثلثی) برای کاهش زمان اجرای هر جستجو از ..... به ۱ می باشیم.

به هر حال دو عامل جبران وجود دارد. اول ، بدلیل آنکه لیست رکوردها لازم به مرتب شدن نیستند ، هر دوی عملیاتهای درج و حذف (extraction) تقریبا به سادگی جستجوکه به زودی مشاهده می کنیم می باشد. دوم ، هر کدام از گره های ما بسیار ساده بوده و در نتیجه به طور قابل ملاحظه ای سریعتر از پردازنده ترتیبی با نیاز به محاسبه آدرس محل مقایسه بعدی ، خواندن اطلاعات از حافظه و سپس تنظیم آدرسهای جدید محدوده جستجو برای هر مقایسه می باشد.

حذف رکورد زمانی که هیچ کلید تکراری وجود ندارد یا حذف نمودن جهت حذف تمام رکورد ها با کلید y داده شده در نظر گرفته شده باشد سر راست می باشد .دستورالعمل delete (y) به تمام گره ها برگ پخش می شود با آن گره های برگ آنهایی که ارزش کلیدش برابر y می باشد به عنوان رکورد delete بر چسب می خورد . باز پس گیری فضای اشغال شده توسط چنین رکورد هایی در قسمت بحث و گفتگو insertخواهد آمد. جستجو های جدید می تواند بلافاصله وارد درخت در ادامه دستور حذف گردد چرا که همزمان با اینکه دستورات جستجو به گره های برگ می رسند حذف نیز در حال صورت گرفتن است.

هر دو دستور extractmin و extractmax باید در دو فاز انجام شود اول مکان گره برگ که نگهدارنده مقدار کلید کمترین یا بیشترین است باید شناخته شود . به محض اینکه نتیجه از گره خروجی ریشه آماده شد یک دستور العمل برای استخراج رکورد درج می شود بنابراین تاخیر این دستورالعمل ها دو برابر بزرگتر شود . علاوه بر این هیچ دستور جستجویی اجازه ورود به درخت را قبل از شروع فاز دوم را ندارد.

درج رکورد نیز سر راست است اگر کلید های تکراری مجاز نباشند و ما مطمئن نباشیم که رکوردی با مقدار کلید y هنوز موجودنیست ما ممکن است دستورinsert(y,z) را قبل از دستور delete (y) بیاوریم .تفاوت اصلی بین دستور insert و قبلی آن است که دستورالعمل تمام به گره های برگ پخش نمی شود چون ما نمی خواهیم هر گره برگ خالی رکورد جدیدی را درج کنند در عوض به مکانیزمی نیازمندیم که بتوانیم به صورت انتخابی دستورالعمل insert را به یک و فقط یک گره خالی مسیر دهی نماییم.

یک مکانیزم ساده برای مسیردهی یک دستورالعمل insert به شرح زیر است: فرض کنید که هر گره غیر برگ دو شمارنده که بیانگر تعداد گره های خالی را در زیر درختهای چپ و راست خود به ترتیب نگهداری می کند باشد وقتی دستور insert بوسیله یک گره غیر برگ دریافت می شود آن به قسمت یکی از زیر درخت ها با شمارش فضای آزاد غیر صفر (شاید به یکی با شمارنده بزرگ تر در صورتی که هر دو غیر صفر یا همیشه به سمت زیر درخت سمت چپ در صورت امکان) فرستاده می شود شمارنده زیر درخت مربوطه بوسیله ۱ کاهش می یابد شکل ۸,۲ مثالی را با ۸ برگ از دستگاه درختی با سه برگ خالی و شمارنده مقدار فضای خالی که مربوط به هر گره دایره ای را نشان می دهد. ما در شکل ۸,۲ میبینیم که دستور insert به بچه سمت چپ گره ریشه فرستاده می شود شماره سمت چپ گره ریشه از یک به صفر کاهش می یابد.

تنها مسئله باقی مانده آن است که نشان دهیم چگونه مقادیر شمارنده در زمان حذف یک رکورد به روز گردد. زمانی که دستور delete از درخت بالایی به پایین فرستاده می شود هنوز مشخص نشده که کدام گره برگ باید حذف را اجرا کند و در نتیجه کدام شمارنده ها باید

افزایش یابند با این حال اگر نیازمند باشیم که ID برگ حذفی در ریشه خروجی آماده گردد دستور دوم می تواند به داخل دستگاه برای تنها هدف به روزرسانی شمارنده فضای آزاد درج گردد. این تاخیر کم بین رکورد درحال حذف شدن و فضایی که دوباره قابل استفاده می شود مسئله جدی نمی باشد مگر اینکه ماشین درختی تقریبا پر حتی این در این مورد اخیر جریمه سرعت پرداخت نمی گردد مگر اینکه یک درج بلافاصله بعد از دستور deleteنیاز گردد.

یک نوع جالب از ماشین درخت برای عملیات های لغتنامه از ساختار داده درخت دودویی سیستولیک که درآن ریشه هر زیر درخت شامل کوچکترین مقدار میانی و بزرگترین مقدار از کلید در آن زیر درخت (شکل ۸،۳)می باشد استفاده می کند وقتی یک زیر درخت تعداد فردی از کلید ها را نگهداری می کند ریشه آن شامل سه مقدار و زیر درختش شامل تعداد مساوی از کلیدها می باشد. در غیر این صورت توسط conrention زیر درخت چپ شامل یک کلید کمتر می شود این اطلاعات در ریشه به صورت یک پرچم تک بیتی نگهداری می شود به طوری که عنصر درج شده بعدی به سمت زیر درخت چپ رانده شده و بیت flip شده برای نشان دادن تعادل کامل می باشد. با این ساختار داده کوچکترین بزرگترین و متوسط مقادیر ذخیره شده به طور سریعتری در یک چرخه ساعت از طریق گره ریشه قابل دسترس خواهد بود تعیین جزییات الگوریتم ها به عنوان تمرین قرار داده شده است.

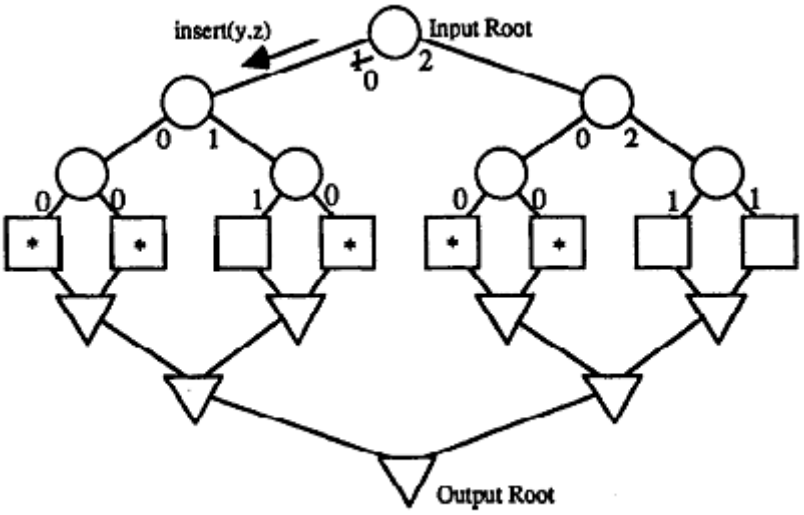

Figure 8.2. Tree machine storing five records and containing three free slots.

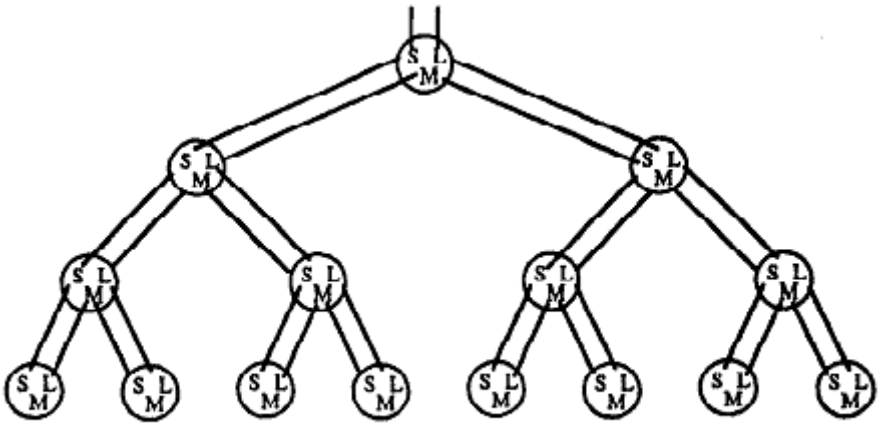

Figure 8.3. Systolic data structure for minimum, maximum, and median finding.

۸,۳ محاسبات پیشوندی موازی

محاسبات پیشوندی موازی در بخش ۲,۱ با چندین الگوریتم که پس از آن در بخش های ۲,۳ (آرایه خطی)۲,۴ (درخت دودویی) ۲,۵ (مش دو بعدی) ۵,۴ (PROM) ارایه شد تعریف گردید. دراین جا ما از یک فرمول جایگزین از نظر recarrnces خطی استفاده می نماییم. عملگر را به عنوان عملگر دودویی انجمن بر روی s در نظر بگیرید به عنوان مثال $(x \otimes y) \otimes z = x \otimes (y \otimes z)$ برای تمام $x, y, z \in S$ با توجه به یک دنباله از مقادیر ورودی $x_1, x_2, \ldots, x_n$حل عود خطی برابر $S_i = S_{i-1} \otimes x_i$به طوری که S_0 عنصر خنثی اپراتور ⊗(به عنوان مثال ۰ برای اضافه ۱ برای ضرب) می باشد توجه داشته باشید که i امین مقدار خروجی برابر $S_i = x_1 \otimes x_2 \otimes \ldots \otimes x_i$

فرمول عود خطی از محاسبه پیشوندی بلافاصله پیاده سازی مدار تربیتی نشان داده شده در شکل ۸,۴ (چپ) را پیشنهاد می دهد یک ورودی برای مدار آماده شده و یک خروجی در هر چرخه ساعت تولید می شود. بدیهی است دوره ساعت باید به اندازه کافی طولانی باشد که تاخیر انتشار سیگنال بدترین حالت در محاسبه مداری ⊗ علاوه بر نگهداری چفت و زمان راه اندازی accommodate شود. شکل ۲,۱ گراف وابستگی محاسبه یا گراف جریان سیگنال برای این الگو را در صورتی که ما از دوره کلاک به عنوان واحد زمانی استفاده کنیم را نشان می دهد. یک مدار تابع لوله ای شده q مرحله کمکی در افزایش سرعت محاسبات پیشوندی نمی کند حتی اگر زمان دوره ای کلاک هم اکنون بسیار کوتاه باشد یک ورودی می تواند بعد از هر q تیک کلاک پشتیبانی می شود. در حقیقت بخاطر سربار اضافه شده برای چفت کردن نتایج میانی واحد تابع لوله ای شده شکل ۸,۴ (راست) در واقع محاسبات را کند می نماید.

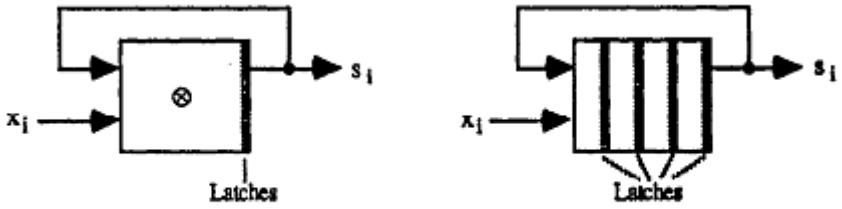

Figure 8.4. Prefix computation using a latched or pipelined function unit.

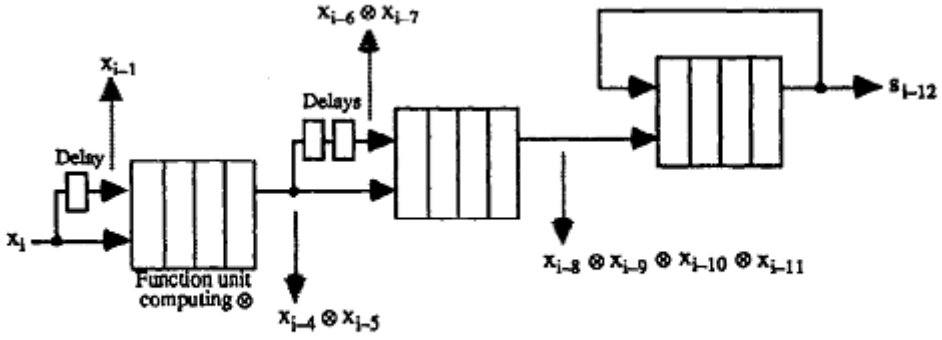

Figure 8.5. High-throughput prefix computation using a pipelined function unit.

شکل ۸,۵ چگونگی کمک موازی سازی بهبود نفوذ برای یک ورودی یا نتیجه در هر تیک کلاک را نشان می دهد. عمومیت دادن این الگو برای استفاده واحد تابع لوله ای شده با مراحل بیشترکه منجر به سرمایه گذاری بیشتر در سخت افزار برای توان عملیاتی بالاتر آسان است.

با این حال محدودیت فراتری وجود دارد که گذر دهی نمی تواند بال استفاده از این الگو بهبود یابد. یعنی اگر سر بار چفت کردن برای هر مرحله لوله c باشد آنگاه گذردهی هرگز نمی تواند از $1/c$ بون توجه به اینکه چندین واحد لوله ای شده استفاده اند یا چگونه مرتب شده اند تجاوز کند.

در بخش ۸,۴ ما بعضی از متد ها برای طراحی سزیعتر مدارات محاسباتی پیشوندی را مرور می کنیم محاسبات پیشوندی کاملا مهم هستند. ما بعضی از برنامه های کاربردی شان را در بخش ۲,۴ بحث می کنیم بسیاری دیگر از برنامه های کاربردی جذاب توسط Lakshmivarahan و Dhall بررسی شده اند.

۸,۴ شبکه های پیشوندی موازی

به خاطر اختصار ما طراحی مدارات برای محاسبه جمع های پیشوندی موازی با ورودی های عدد صحیح بدون علامت یعنی اپراتور $\otimes$ برای اضافه کردن عدد صحیح بدون علامت گرفته شده بحث خواهیم کرد.مدارات نتیجه شده که از جمع کننده (adder) دو ورودی ساخته شده به عنوان شبکه های مجموع پیشوندی (موازی) اشاره می شوند . با جایگزینی جمع کننده دو ورودی در این شبکه هابا بلاک هایی که اپراتور $\otimes$ را محاسبه می کند به طوری کلی شبکه پیشوندی موازی را yield می کند.

شباهت های بسیاری بین شبکه های مرتب سازی و شبکه های مجموع پیشوند وجود دارد که در فصل ۷ بحث شد یک جمع کننده دودویی دو ورودی دارای تقریبا پیچیدگی یکسانی همانند مقایسه گر دو ورودی می باشد مانند مقایسه گر ها این جمع کننده ها می توانند با ورودی های موازی یا با ورودی های بیت سریال پیاده سازی شوند. در مورد اخیر ترتیب ورودی باید LSB-First به عنوان مخالف با MSB-first برای مقایسه گرهای دو ورودی انتخاب شود. توجه کنید که ورودی های موازی استفاده شوند. یک جمع کننده ripple-cary انجام خواهد داد ، چراکه تاخیر rippling در یک مجموعه آبشاری جمع کننده ها اضافه نخواهد شد. Rippling در جمع کننده پایین دست می تواند به محض اینکه LSB جمع از جمع کننده قبلی در دسترس قرار گرفت شروع شود.

استراتژی های متعددی برای سنتز یک شبکه مجموع پیشوندی می تواند مورد استفاده قرار گیرد. متدهای دوتایی تقسیم و غلبه که در ارتباط با PRAM در بخش ۵,۴ بحث شد و در شکلهای ۵,۷ و ۵,۸ نشان داده شد می تواند پایه و اساس شبکه های مجموع پیشوندی سخت افزاری را تشکیل دهد. شکلهای ۸,۶ و ۸,۷ شبکه های حاصل را نمایش می دهد.تجزیه و تحلیل بخش ۵,۴ می تواند برای استنباط تاخیر $T(n) = 2\log_2 n - 1$ و $T(n) = \log_2 n$ برای این مدارات به طوری که واحد زمان میزان تاخیر یک جمع کننده دو ورودی است مورد استفاده قرار گیرد. عود هزینه برای شبکه های شکلهای ۸,۶ و ۸,۷ به ترتیب برابر $\mathrm{C(n) = C(n/2) + n - 1 = 2n - 2 - \log_2 n}$ و $\mathrm{C(n) = 2C(n/2) + n/2 = (n/2)\log_2 n}$ می باشد.

دو طراحی در شکلهای ۸,۶ و ۸,۷ یک هزینه سرعت تجاری روشن را ارائه می دهد. طراحی دوم سریعتراست ($\log_2 n$ به عنوان مخالف سطوح $2\log_2 n - 1$) اما همچنین بسیار گران می باشد [جمع کننده $(n/2)\log_2 n$ مخالف بلوکهای جمع کننده $2n - 2 - \log_2 n$]. مسئله دیگر در مورد طراحی دوم آناست که آنمنجر به نیاز به پهنای خروجی بزرگ درصورت اجرا به صورت مستقیم می گردد. ما بزودی میبینیم که طراحی های متوسط ،با هزینه ها و تاخیرها که بین دو افراط بالا سقوط می کند ، همچنان غیر ممکن است.

طراحی نشان داده شده در شکل ۸,۶ به عنوان گراف پیشوندی موازی Brent-kung شناخته می شود. ۱۶ ورودی نمونه این گراف در شکل ۸,۸ نشان داده شده است. توجه کنید که حتی اگر گراف شکل ۸,۸ ظاهرا ۷ سطح دارد ، سطوح ۴ و ۵ از بالا هیچ وابستگی داده ای نداشته و به این ترتیب به یک سطح سیگنال از سیگنال تاخیر دلالت دارد. به طور کلی ، یک n ورودی گراف پیشوندی موازی Brent-kungدارای تاخیر $2\log_2 n - 2$ سطح و هزینه $2n - 2 - \log_2 n$ بلوک می باشد.

شکل ۸,۹ گراف پیشوندی موازی kogge-stoneرا نشان داده که دارای تاخیر همسان همانطور که طرح در شکل ۸,۷ نشان می دهد می باشد. اما بوسیله توزیع محاسبات مورد نیاز از مسئله پهنای خروجی اجتناب می کند. یک گراف پیشوندیموازی Kogge-stone دارای n ورودی دارای تاخیر $\log_2 n$ سطحی و هزینه $n\log_2 n - \mathrm{n} + 1$ بلاک را دارد. گراف پیشوندی موازیkogge-stone سریعترین اجرای ممکن از محاسبات پیشوندی موازی را در صورتی که فقط بلاک دو- ورودی مجاز باشد را نشان می دهد. به هر حال ، هزینه اش می تواند برای nهای بزرگ ، از نظر هر دو بلاک و تراکم سیم کشی بین بلاکها گران باشد.

طرحهای شبکه پیشوندی موازی دیگر همچنین ممکن است. برای مثال ، پیشنهاد شده است که روشهای Brent-kung و kogge-stone شکلهای ۸,۸ و ۸,۹ قابل ترکیب شده برای طرحهای ترکیبی [sug190] می باشد. شکل ۸,۱۰ مثالی را نشان می دهد بطوری که میانه چهار تا از شش سطح در طرح شکل ۸,۸ (در اصل انجام محاسبات پیشوندی موازی هشت ورودی) بوسیله شبکه Kogge-stone هشت ورودی جایگزین شده است. طرح نتیجه دارای ۵ سطح و ۳۲ بلوک می باشد ، که بین طرحهای خالص Brent-kung(6,26) و Kogge-stone(4,47) جایگزین می شود.

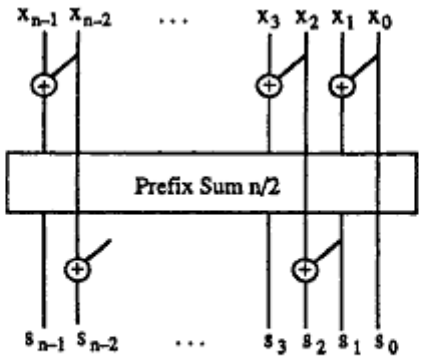

Figure 8.6.Prefix sum network built of one *n*/2-input network and *n* – 1 adders.

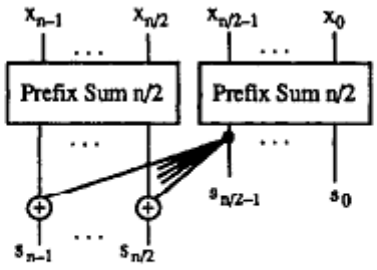

Figure 8.7. Prefix sum network built of two *n*/2-input networks and *n*/2 adders.

به طور کلی ، اگر سطح Brent-kung به همراه طرح kogge-stone ، n/2 ورودی استفاده شود ،تاخیر و هزینه شبکه ترکیبی به ترتیب $\log_2 n + 1$و $(n/2)\log_2 n$می گردد. طرح نتیجه نزدیک به کمترین از نظر تاخیر می شود(فقط یکی بیشتر از Kogge-stone) اما هزینه تقریبا نیمی بیشتر است.

تئوری گرافهای پیشوندی موازی به خوبی توسعهیافتهاست. تعداد زیادی مرزهای نظری و طرحهای واقعی با محدودیتهای مختلف در پهنای ورودی/خروجی و بهبود معیارهای مختلف ازنظر تاخیر و هزینه شبکه وجود دارد. برای بحث جزئی تر بیشتر ،lakshmivarahan و Dhall ([lak94] ، pp۲۱۱-۱۳۳) را ببینید.

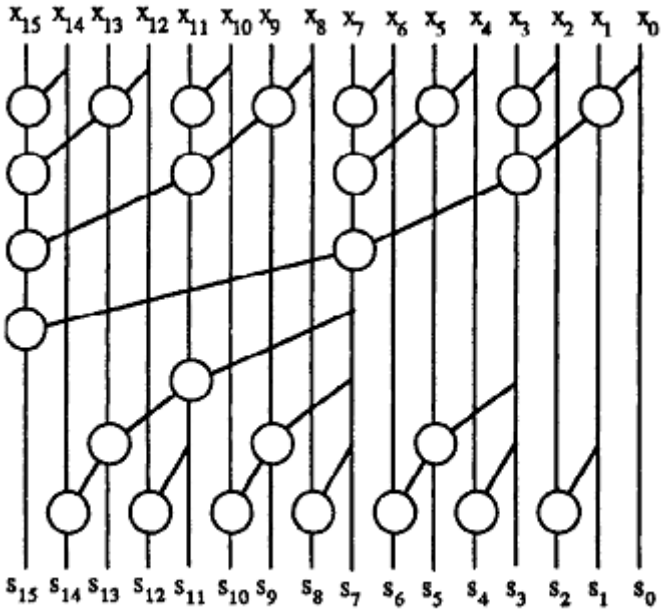

Figure 8.8. Brent–Kung parallel prefix graph for 16 inputs.

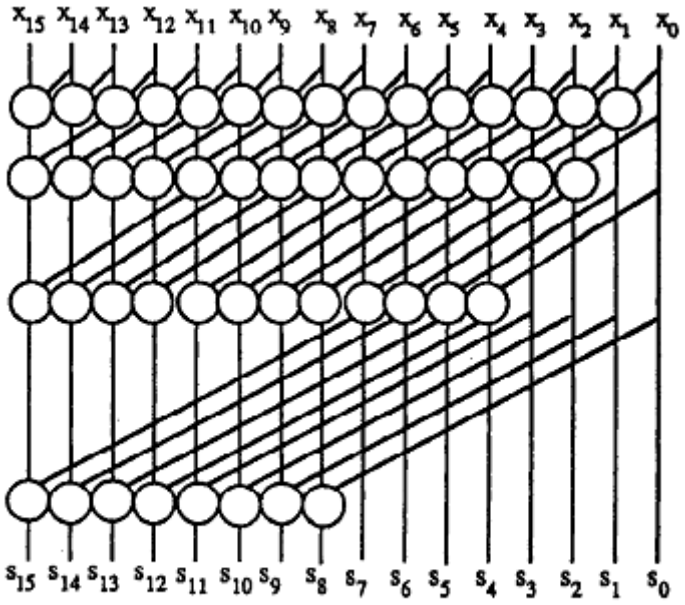

Figure 8.9. Kogge–Stone parallel prefix graph for 16 inputs.

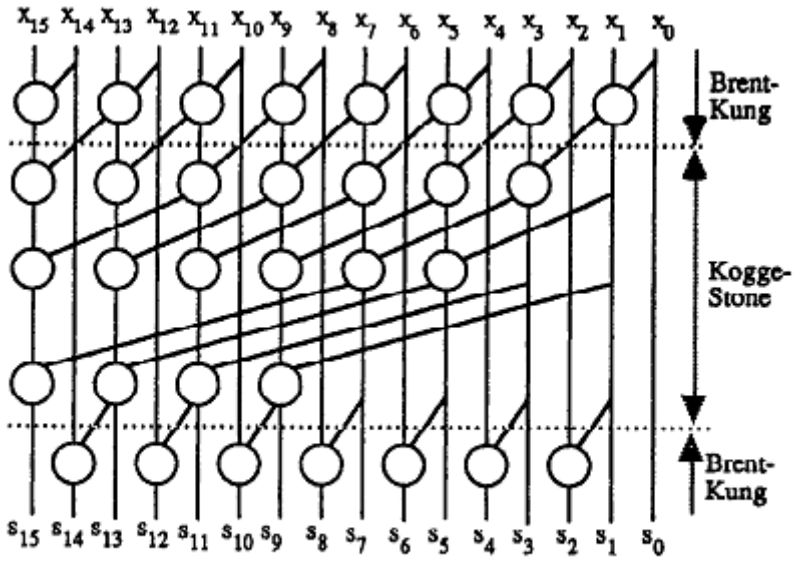

Figure 8.10. A hybrid Brent–Kung/Kogge–Stone parallel prefix graph for 16 inputs.

۸,۵ تبدیل فوریه گسسته

تبدیل فوریه گسسته(DFT) به صورت زیر تعریف می شود. با توجه به توالی $x_0,x_1,\ldots,x_{n-1}$ ، دنباله $y_0,y_1,\ldots,y_{n-1}$ را با توجه به

$$y_i = \sum_{j=0}^{n-1} \omega_n^{ij} x_j$$

محاسبه کنید.

DFT به صورت ماتریس به عنوان $y = F_n^x$ که برای تندنویسی

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)^2} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix}$$

ابراز شده است.

در عبارت بالا ، W_n ریشه nام اصلی واحد به طور مثال $w_n^n = 1$ و $w_n^j \neq 1$ برای $1 \leq j < n$ می باشد. به عنوان مثال ، $w_3 = -1/2 + i\sqrt{3}/2$ و $W_4 = i$ بهطوری که $i = \sqrt{-1}$ می باشد. عبارت زیر به سادگی مشتق می شود.

$$\omega_n = e^{2\pi i/n}$$

$$\omega_n^j = e^{2\pi ji/n} = \cos(2\pi j/n) + i\sin(2\pi j/n)$$

معکوسDFT قابل استفاده برای بازیابی دنباله x ،با توجه به دنباله y داده شده می باشد. این می تواند نشان داده شود که معکوس DFT اساسا دارای محاسبات همسان با DFT می باشد. آن عبارت است از

$$x_i = \frac{1}{n}\sum_{j=0}^{n-1}\omega_n^{-ij}y_j$$

توجه داشته باشید که n تا پردازنده می تواند DFT را در o(n) مرحله بدون هیچ ارتباطی ، در صورتی که jامین پردازنده y_j را محاسبه کند، محاسبه نماید. همچنین ،با توجه به فرمول ماتریس DFT ، هر الگوریتم ضرب ماتریس برداری می تواند برای محاسبه DFT ها استفاده گردد. با این حال ، ساختار ویژه F_n می تواند برای درست کردن یک الگوریتم بسیار سریعتر مبتنی بر نمونه تقسیم و غلبه مورد استفاده قرار بگیرد. الگوریتم حاصل ، که در زیر شرح داده شده ، به عنوان الگوریتم تبدیل فوریه سریع (FFT) شناخته شده است.

اجازه دهید مجموع DFT را به شروط شاخص فرد و زوج پارتیشن بندی کنیم.

$$y_i = \sum_{j=0}^{n-1}\omega_n^{ij}x_j = \sum_{j\ \text{even}\ (2r)}\omega_n^{ij}x_j + \sum_{j\ \text{odd}\ (2r+1)}\omega_n^{ij}x_j$$

$$= \sum_{r=0}^{n/2-1}\omega_{n/2}^{ir}x_{2r} + \omega_n^i\sum_{r=0}^{n/2-1}\omega_{n/2}^{ir}x_{2r+1}$$

به طور که ما از شناسه $w_{n/2} = w_n^2$برای بازنویسی دو شرط جمع استفاده کرده ایم. دو شرط در عبارت قبلی برابر n/2 نقطه در DFTها که به صورت

$$u = F_{n/2}\begin{bmatrix} x_0 \\ x_2 \\ \vdots \\ x_{n-2} \end{bmatrix} \quad v = F_{n/2}\begin{bmatrix} x_1 \\ x_3 \\ \vdots \\ x_{n-1} \end{bmatrix}$$

قابل نوشتن است می باشد. آنگاه

$$y_i = \begin{cases} u_i + \omega_n^i v_i & 0 \le i < n/2 \\ u_{i-n/2} + \omega_n^i v_{i-n/2} & n/2 \le i < n \end{cases} \quad \text{or} \quad y_{i+n/2} = u_i + \omega_n^{i+n/2}v_i$$

از این رو ، الگوریتم FFT ، n نقطهای شامل اجرای دو FFT ، n/2 نقطه ای مستقل می باشد و آنگاه نتایج خود را با استفاده از nتا عملیات ضرب –جمع ترکیب می کند می باشد. پیچیدگی زمانی ترتیبی FFT در نتیجه مشخص نمودن بوسیله بازگشت می باشد.

$$T(n) = 2T(n/2) + n = n\log_2 n$$

در صورتی که دو زیرمسئله n/2 نقطهای قابل حل در موازی باشند وعملیاتهای n تایی ضرب- جمع نیز همزمان باشند ، با ورودی های مورد نیازشان که به گره های محاسبهای فورا منتقل شده اند ، آنگاه بازگشت برای پیچیدگی زمانی موازی برابر

$$T(n) = T(n/2) + 1 = \log_2 n$$

در دو حالت ،واحد زمانی ، تاخیر انجام یک عملیات ضرب – جمع می باشد.

قبل از بحث در مورد مدارات سخت افزاری برای پیاده سازی الگوریتم FFT ، بررسی بعضی برنامه های کاربردی DFT به حساب چند جملهای آموزنده می باشد. ابتدا ، توجه کنید که DFT می تواند به صورت چند جمله ای ارزیابی شود:

با توجه به چند جمله ای $f(x) = c_{n-1}x^{n-1} + \ldots + c_1x + c_0$

حساب کردن $f(\omega_n^0), f(\omega_n^1), \ldots, f(\omega_n^{n-1})$

در فرم ماتریسی

$$\begin{bmatrix} f(\omega_n^0) \\ f(\omega_n^1) \\ \vdots \\ f(\omega_n^{n-1}) \end{bmatrix} = F_n \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{bmatrix}$$

در یک روش مشابه ، معکوس DFT قابل استفاده برای درج چند جمله ای می باشد:

$$\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{bmatrix} = F_n^{-1} \begin{bmatrix} f(\omega_n^0) \\ f(\omega_n^1) \\ \vdots \\ f(\omega_n^{n-1}) \end{bmatrix} \quad \text{with } F_n^{-1}[i,j] = \frac{1}{n}\omega_n^{-ij} = \frac{1}{n}(F_n[i,j])^{-1}$$

مسئله ضرب دو چند جمله ای f و g برای بدست آوردن حاصل شان h ، به طوری که

$$g(x) = a_{n'-1}x^{n'-1} + \cdots + a_1 x + a_0$$

$$h(x) = b_{n''-1}x^{n''-1} + \cdots + b_1 x + b_0$$

$$f(x) = g(x) \times h(x) = c_{n-1}x^{n-1} + \cdots + c_1 x + c_0 \quad \text{with} \quad n = n' + n'' - 1$$

آن است که ، محاسبه c_js با توجه به a_js و b_js ، قابل تبدیل به سه DFT ، n نقطه ای و $n = n' + n'' - 1$

مجموعه ضرایب به شرح ذیل می باشد:

۱ - ارزیابی g(x) و h(x) در nامین ریشه واحد (دو DFT ، n نقطه ای).

۲ - ارزیابی $f(\omega_n^j) = g(\omega_n^j) + h(\omega_n^j)$برای تمام j (مجموعه ضرایب n).

۳ - درمیان عبارات دیگر جا دادن برای پیدا کردن c_js (یک معکوس DFT ، n نقطه ای)

بنابراین ، $T_{poly-mult} = 3T_{DFT} + 1$اگر ما از الگوریتم FFT به تعداد O(logn) بار استفاده کنیم ، زمان کل برای ضرب چند جمله ای O(logn) می شود. پیچیدگی دو دنباله n′ عنصری a و b ، تعریف شده به عنوان محاسبه دنباله nعنصری c با $C_j = a_j b_0 +$ $a_{j-1}b_1 + \cdots + a_0 b_j$برای $0 \le j \le 2n' - 2$ با ضرب چند جمله ای یکسان است.

در نهایت ، ضرب عدد صحیح نزدیک به ضرب چند جملهای مرتبط با آن ،ودرنتیجه DFT میباشد.

به طورمستقیم ضرب K بیتی صحیح مدارات نیازمند تاخیر O(log k) و هزینه $O(k^2)$ (k^2 بیت افزایش دهنده multiplier با گیت AND ، و به دنبال آن درخت جمع کننده ذخیره کری و جمع کننده سریع زمان لگاریتمی). یک الگوریتم ضرب O(log K) باری عدد صحیح برای K بیت عدد ، پردازنده های $O(k \log^2 k \log\log k)$ سطحی بیت رااستفاده می کند، که توسط Leighton ارائه شده. مدار ضرب حاصل قابل استفاده برای ضرب O(log K) جفت از اعداد صحیح K بیتی در زمان O(log k)از طریق لوله ای می باشد ، درنتیجه منجر به هزینه O(k log k log logk) برایهر ضرب ، که نزدیک به مطلوب است می شود.

۸,۶ معماری موازی برای FFT

شکل ۸,۱۱ (چپ) شبکه FFT هشت نقطه ای را که مستقیما از الگوی محاسبه تقسیم و غلبه بخش ۸,۵ مشتق شده است را نشان می دهد. هر گره دایره ای یک محاسبه ضرب-جمع را محاسبه می کند.اثبات مقایسه ای که این شبکه در واقع انجام می دهد ، FFt ای که می تواند به راحتی ساخته شود را محاسبه می کند. فرض کنید که مقادیر uj و vj موجودند ، نتیجه FFT ، n/2 نقطه ای با ورودی های زوج و فرد شاخص بندی شده ، در بالاترین و پایین ترین بخشهای مدار به دست می آیند ، همانطور که در شکل ۸,۱۱ نمایش داده شده است. آنگاه ، آن بسیار ساده است که ببینید آخرین مرحله از مدار فقط اتصال به راست را برای اتمام محاسبه با اجرای nتا جمع-ضرب مورد نیاز انجام داده است.

با مرتب سازی دوباره گره های مدار FFT یمان ، که به عنوان شبکه پروانه شناخته شده است ، ما به یک نمایش معادل همانطور که در شکل ۸,۱۱(راست) نشان داده شده رسیده ایم. این نمایش به عنوان شبکه Shuffle-exchange شناخته شده است. این شبکه و دلیل نامگذاریش در بخش ۱۵,۵ بحث خواهد شد. در حال حاضر ، ما آن را به سادگی نمایش معادل شبکه FFT که مزایای اتصالات یکسان بین مراحل مختلف گره ها را ارائه می دهد را می بینیم. در شبکه FFt اصلی ، ورودی ها به شاخصهای زوج و فرد تقسیم شده و برای نیمه بالا و پایین مدار به همان ترتیب ، که خروجی ها در یک ترتیب شاخص صعودی بدست آمده اند آماده می شوند. در نسخه معادل Shuffle-exchange ورودی ها به صورت از بالا به پایین ، با خروجی های بدست آمده در ترتیب عکس بیتی مرتب شده اند.

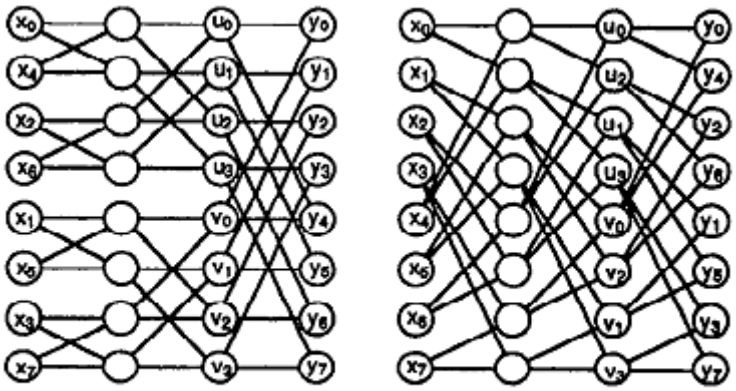

Figure 8.11. Butterfly network for computing an eight-point FFT.

این بدان معنی است که ترتیب شاخصهای عناصر خروجی توسط معکوس نمایشهای باینریشان مرتب شده اند. برای مثال ، نمایش باینری عدد ۴ می شود ۱۰۰ ، که عکس آن ۰۰۱ می شود. در نتیجه ، y4 خروجی دوم از بالا می باشد.

شبکه های FFt بسیار موثرتری ممکن است. برای مثال ، Yeh و parhami [YEH96] کلاسی از شبکه های FFT که بسیار مدولارتر از آنهایی که در شکل ۸,۱۱ هستندرا نمایش می دهد که در نتیجه در اجرای VLSI دارای هزینه کمتری میباشد. مدار Shuffle-exchange شکل ۸,۱۱ (راست) می تواند تا حدودی با حذف بعضی از اتصلات همانطور که در شکل ۸,۱۲ (چپ) نشان داده شده ساده گردد. فلشها در شکل ۸,۱۲ مسیر حرکت داده (از طریق گره های میانی) به جای لینک های از دست رفته را نشان می دهند.

شبکه پروانه ، و انواع آن که در بالا بحث شد ، می تواند برای nهای بزرگ کاملا پیچیده گردد. در این مورد ، مقرون به صرفه ترین پیاده سازی ممکن است دنبال شود. یک راه توسعه مدار شکل ۸,۱۲ (چپ) در جهت افقی است ، به طوری که یک گره ضرب – جمع در مراحل زمانی پی در پی ، تابع تمام گره ها واقع شده در یک ردیف را انجام دهد. مدار حاصل در شکل ۸,۱۲ (راست) نمایش داده شده است. در اینجا ، یک ستون از گره های ضرب –جمع به طور متناوب به عنوان ستونهای مختلف از گره ها در دیاگرام در سمت چپ ، با نتایج محاسبات جزئی که از یک ستون به بعدی ذخیره شده در ثبات ها یا چفتها برای استفاده در مرحله زمانی جاری عمل می کند. این روش هزینه شبکه را از O(nlogn) به O(n) بدون افزایش قابل توجه تاخیرش کاهش می دهد.

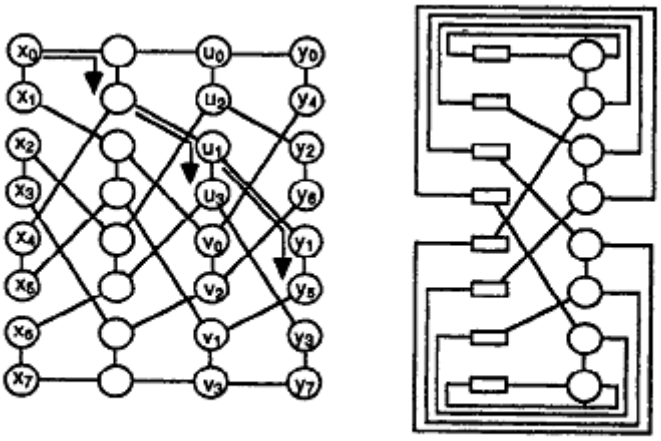

Figure 8.12. FFT network variant and its shared-hardware realization.

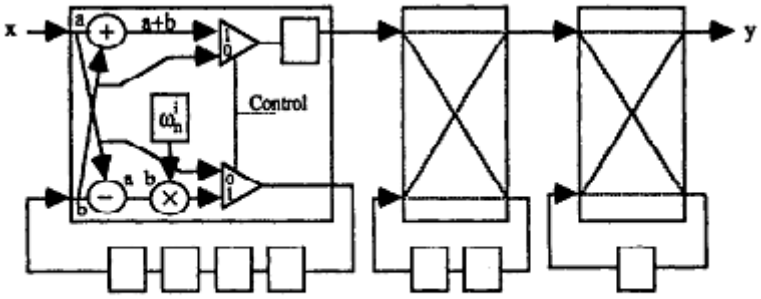

Figure 8.13. Linear array of $\log_2 n$ cells for n-point FFT computation.

حتی نتایج پیاده سازی اقتصادی تر در صورتی که دیاگرام به جای افقی ، عمودی توسعه یابد بدست می آید. شکل ۸,۱۳ نتیجه مدار آرایه خطی را با بازخورد شیفت ریجستر ها درج شده برای حفظ زمانبندی مناسب گره های ورودی مختلف [kwai96] را نشان می دهد.

کشف جزئیات مربوط به چگونگی کار این طرحها به عنوان تمرین باقی می ماند.