تحلیل الگوریتم شاخه و قید موازی آسنکرون

Asynchronous Parallel Branch and Bound Algorithm

۱- خلاصه:

در این مقاله توضیحی درباره کامپیوترهای موازی می‌دهیم و بعد الگوریتمهای موازی را بررسی می‌کنیم. ویژگیهای الگوریتم branch & bound را بیان می‌کنیم و الگوریتمهای b&b موازی را ارائه می‌دهیم و دسته‌ای از الگوریتمهای b&b آسنکرون برای اجرا روی سیستم MIMD را توسعه می‌دهیم. سپس این الگوریتم را که توسط عناصر پردازشی ناهمگن اجرا شده است بررسی می‌کنیم.

نمادهای perfect parallel و achieved effiency را که بطور تجربی معیار مناسبی برای موازی‌سازی است معرفی می‌کنیم زیرا نمادهای قبلی speed up (تسریع) و efficiency (کارایی) توانایی کامل را برای اجرای واقعی الگوریتم موازی آسنکرون نداشتند. و نیز شرایی را فراهم کردیم که از آنومالیهایی که به جهت موازی‌سازی و آسنکرون بودن و یا عدم قطعیت باعث کاهش کارایی الگوریتم شده بود، جلوگیری کند.

۲- معرفی:

همیشه نیاز به کامپیوترهای قدرتمند وجود داشته است. در مدل سنتی محاسبات، یک عنصر پردازشی منحصر تمام taskها را بصورت خطی (Seqventia) انجام میدهد. به جهت اجرای یک دستورالعمل داده بایستی از محل یک کامپیوتر به محل دیگری منتقل

می‌شد، لذا نیاز هب کامپیوترهای قدرتمند اهمیت روز افزون پیدا کرد. یک مدل جدید از محاسبات توسعه داده شد، که در این مدل جدید چندین عنصر پردازشی در اجرای یک task واحد با هم همکاری می‌کنند. ایده اصل این مدل بر اساس تقسیم یک task به subtaskهای مستقل از یکدیگر است که می‌توانند هر کدام بصورت parallel (موازی) اجرا شوند. این نوع از کامپیوتر را کامپیوتر موازی گویند.

تا زمانیکه این امکان وجود داشته باشد که یک task را به زیر taskهایی تقسیم کنیم که اندازه بزرگترین زیر task همچنان به گونه‌ای باشد که باز هم بتوان آنرا کاهش داد و البته تا زمانیکه عناصر پردازشی کافی برای اجرای این sub task ها بطور موازی وجود داشته باشد، قدرت محاسبه یک کامپیوتر موازی نامحدود است. اما در عمل این دو شرط بطور کامل برقرار نمی‌شوند:

اولاً: این امکان وجود ندارد که هر taskی را بطور دلخواه به تعدادی زیر taskهای مستقل تقسیم کنیم. چون همواره تعدادی زیر task های وابسته وجود دارد که بایستی بطور خطی اجرا شوند. از اینرو زمان مورد نیاز برای اجرای یک task بطور موازی یک حد پایین دارد.

دوماً: هر کامپیوتر موازی که عملاً ساخته می‌شود شامل تعداد معینی عناصر پردازشی (Processing element) است. به محض آنکه تعداد taskها فراتر از تعداد عناصر پردازشی برود، بعضی از sub task ها بایستی بصورت خطی اجرا شوند و بعنوان یک فاکتور ثابت در تسریع کامپیوتر موازی تصور می‌شود.

الگوریتمهای B&B مسائل بهینه سازی گسسته را به روش تقسیم فضای حالت حل می‌کنند. در تمام این مقاله فرض بر این است که تمام مسائل بهینه سازی مسائل می‌نیمم کردن هستند و منظور از حل یک مسئله پیدا کردن یک حل ممکن با مقدار می‌نیمم است. اگر چندین حل وجود داشته باشد، مهم نیست کدامیک از آنها پیدا شده.

الگوریتم B&B یک مسئله را به زیر مسئله‌های کوچکتر بوسیله تقسیم فضای حالت به زیر فضاهای (Subspace) کوچکتر، تجزیه می‌کند. هر زیر مسئله تولید شده یا حل است و یا ثابت می‌شود که به حل بهینه برای مسئله اصلی (Original) نمی‌انجامد و حذف می‌شود. اگر برای یک زیر مسئله هیچ کدام از این دو امکان بلافاصله استنباط نشود، آن زیر مسئله به زیرمسئله‌های کوچکتر دوباره تجزیه می‌شود. این پروسه آنقدر ادامه پیدا می‌کند تا تمام زیر مسئله‌های تولید شده یا حل شوند یا حذف شوند.

در الگوریتمهای B&B کار انجام شده در حین اجرا به شدت تحت تاثیر نمونه مسئله خاص قرار می‌گیرد. بدون انجام دادن اجرای واقعی الگوریتم این امکان وجود ندارد که تخمین درستی از کار انجام شده بدست آورد. علاوه برآن، روشی که کار باید سازماندهی شود بر روی کار انجام شده تاثیر می‌گذارد. هر گامی که در اجرای الگوریتم b&b ی موازی بطور موفقیت‌آمیزی انجام می‌شود و البته به دانشی است که تاکنون بدست آورده. لذا استفاده از استراتژی جستجوی متفاوت یا انشعاب دادن چندین زیر مسئله بطور موازی باعث بدست آمدن دانشی متفاوت می‌شود پس می‌توان با ترتیب متفاوتی زیر مسئله‌ها را انشعاب داد.

دقت کنید که در یک بدل محاسبه خطی افزایش قدرت محاسبه فقط بر روی تسریع الگوریتم اثر می‌کند وگرنه کار انجام شده همچنان یکسان است.

با این حال اگر قدرت محاسبه یک کامپیوتر موازی با اضافه کردن عناصر پردازشی اضافه افزایش پیدا کند. اجرای الگوریتم b&b بطور آشکاری تغییر می‌کند (به عبارت دیگر ترتیبی که در آن زیر برنامه‌ها انشعاب پیدا می‌کنند تغییر می‌کند). بنابراین حل مسائل بهینه‌سازی گسسته سرسع بوسیله یک کامپیوتر موازی نه تنها باعث افزایش قدرت محاسبه کامپیوتر موازی شده است بلکه باعث گسترش الگوریتمهای موازی نیز گشته است.

۳- کامپیوترهای موازی (Parallel computers):

یکی از مدلهای اصلی محاسبات Control drivenmodel است، در این مدل کاربر باید صریحاً ترتیب انجام عملیات را مشخص کند و آن دسته از عملیاتی که باید به طور موازی اجرا شوند را تعیین کند. این مدل مستقل از عناصر پردازش به صورت زیر تقسیم‌بندی می‌شود:

- کامپیوترهای SISD، که یک عنصر پردازشی وجود دارد و توان انجام فقط یک عمل را در یک زمان دارد.
- کامپیوترهای MIMD، دارای چندین عنصر پردازشی هستند که بطور موازی دستورالعمل‌های متفاوت را روی دیتاهای متفاوت انجام می‌دهند.

– کامپیوترهای SIMD، همه عناصر پردازشی‌شان یک دستور یکسان را در یک زمان بر روی داده‌های متفاوتی انجام می‌دهند. اگر چه امکان پنهان کردن عناصر پردازشی وجود دارد. عنصر پردازشی پنهان شده نتیجه عملی را که انجام داده ذخیره نمی‌کند.

سیستمهای SIMD بر اساس نحوه ارتباط و اتصال عناصر پردازشی به یکدیگر خود به بخشهایی تقسیم می‌شوند: اگر تمام عناصر پردازشی به یکدیگر متصل باشند و از طریق یک حافظه مشترک ارتباط داشته باشند، به آن tightly coupled system گویند.

و اگر عناصر پردازش حافظه مشترک نداشته باشند اما از طریق شبکه‌ای بهم متصل باشند و بروش message passing با هم ارتباط داشته باشند، به آن loosely coupled system گویند.

حافظه مشترک در tightly coupled system ها هم نقطه قوت و هم نقطه ضعف این سیستمها است. امکان به اشتراک گذاشتن راحت و سریع اطلاعات بین عناصر پردازشی مختلف را فراهم می‌کند. ارتباط به عملیات ساده read و wite روی حافظه مشترک خلاصه می‌شود و هر عنصر پردازشی مستقیماً با دیگر عناصر پردازشی ارتباط برقرار می‌کند. با این حال، اگر تعداد عناصر پردازشی متصل به حافظه مشترک افزایش یابد، حافظه مشترک تبدیل به گلوگاه (Bottleneck) می‌شود.

بنابراین تعداد عناصر پردازشی در یک سیستم tightly coupled محدود است. به جهت اینکه تمام عناصر پردازشی بایستی به ان حافظه مشترک متصل باشند، این سیستمها

بصورت کامل از پیش ساخته هستند و امکان اضافه کردن عناصر پردازش به سیستم وجود ندارد.

از طرف دیگر، ارتباط در یک سیستم loosely coupled کند و آهسته است. تبادل پیامها نیاز به زمانی بیش از زمان لازم برای نوشتن یا خواندن از یک حافظه مشترک دارد. این امکان هم وجود دارد که یک عنصر پردازش مستقیماً به عنصر پردازش دیگر که قصد ارتباط دارد متصل نباشد.

در مقابل compactness بودن سیستمهای tightly coupled ، عناصر پردازشی در یک سیستم loosely coupled می‌توانند در تمام نقاط توزیع شوند. لذا فاصله فیزیکی که یک پیام باید طی کند، بیشتر می‌شود. به جهت این حقیقت که عناصر پردازشی برای ارتباط در یک شبکه از یک پروتکل استفاده می‌کنند، lossely coupled system می‌توانند شامل انواع مختلفی از عناصر پردازشی باشند. امکان اضافه کردن عناصر پردازشی اضافه‌تری به سیستم وجود دارد. در حالت کلی عناصر پردازشی خودشان یک کامپیوتر کاملی هستند.

مثالی از سیستمهای loosely coupled، Distributed Processing utilities Package است که بعداً به تفضیل درباره آنها توضیح می‌دهیم.

٤- الگوریتمهای موازی (Parallel Algorithm):

یک الگوریتم موازی شامل sub taskهایی است که باید انجام شود. بعضی از این sub taskها بصورت موازی اجرا می‌شوند، اما گاهی sub taskهایی هم وجود دارد که باید بصورت خطی اجرا شوند. اجرای هر sub task توسط یک پروسس مجزا انجام می‌شود.

از ویژگیهای مهم یک الگوریتم موازی نحوه محاوره این پروسسها، سنکرون بودن و قطعی بودن الگوریتم است. دو پروسس با یکدیگر محاوره (interact) دارند، اگر خروجی یکی از آندو پروسس ورودی دیگری باشد. نحوه محاوره دو پروسس می‌تواند بطور کامل مشخص شده باشد یا نباشد. اگر مشخص شده باشد، این دو پروسس فقط زمانی می‌توانند ارتباط داشته باشند که هر دو مایل به انجام ارتباط باشند. اگر گیرنده هنوز آماده ارتباط نباشد، فرستنده نمی‌تواند اقدامی انجام دهد.

در حین اجرای یک الگوریتم سنکرون تمام پروسسها باید قبل از محاوره با یکدیگر همزمان شوند. سنکرون شدن در اینجا یعنی قبل از آغاز subtask جدید، آنها باید منتظر کامل شدن عمل دیگر پروسسها باشند. وقتی یک الگوریتم آسنکرون اجرا می‌شود، پروسسها لازم نیست که منتظر یکدیگر شوند تا taskهایشان را تمام کنند. البته این امکان وجود دارد که یک الگوریتم آسنکرون تا حدی سنکرون شود.

یک الگوریتم قطعی است اگر هر بار که الگوریتم بر روی ورودی مشابه اجرا شود، نتیجه اجرا یکسان باشد. یعنی دستورالعملهای مشابه به ترتیب مشابه انجام شود. بنابراین اجراهای متوالی از یک الگوریتم همیشه خروجی یکسان دارد در حالیکه در الگوریتمهای غیر قطعی یک تصمیم یکسان خروجیهای متفاوتی دارد. مثلاً خروجی یک تصمیم ممکن است و البته به فاکتورهای محیطی معینی باشد که توسط الگوریتم کنترل نمی‌شود. از اینرو اجراهای پی‌در پی یک الگوریتم غیر قطعی، خروجی‌های متفاوت تولید می‌کند.

غیرقطعی بودن در سیستم tightly coupled هم رخ دهد چون امکان دارد یک پروسس متغیری را از حافظه مشترک در لحظه‌ای بخواند که پروسس دیگر می‌خواهد روی آن متغیر بنویسد. (توجه کنید که در الگوریتمهای آسنکرون، ترتیب عملیات read و write اهمیتی ندارد.)

در سیستم loosely coupled هم غیر قطعی بودن رخ می‌دهد، اگر زمانیکه task بعدی که باید انجام شود توسط یک پروسس وابسته به پیامی از پروسس دیگر باشد.

(در الگوریتمهای آسنکرون، درباره ترتیب ارسال و دریافت پیام و بررسی وجود پیام صحبتی نمی‌شود.)

کاری که توسط الگوریتم موازی انجام می‌شود صرف computation و communication می‌شود. بنابراین پیچیدگی کل یک الگوریتم موازی به شامل پیچیدگی محاسبه و پیچیدگی ارتباط است.

الگوریتمهای سنکرون نسبت به الگوریتمهای آسنکرون کار را بین پروسسها بطور جدی‌تری تقسیم می‌کنند. در یک الگوریتم سنکرون تقسیم کار فقط وابسته به نوع مسئله مورد حل است نه وابسته به نمونه مسئله خاص، قبل از آنکه اجرای الگوریتم شروع شود، معمولاً مشخص است که چگونه کار باید بین پروسسهای موجود تقسیم شود. در حالیکه در الگوریتم آسنکرون تقسیم کار می‌تواند وابسته به نمونه مسئله خاص باشد، بنابراین تقسیم دقیق کار در زمان اجرا مشخص می‌شود.

اجتناب از سنکرون کردن (همزمانی) دو مزیت مهم دارد:

اولاً: در حالتی که taskهای اجرا شونده سایز یکسانی ندارند، عناصر پردازشی می‌توانند کارایی بالاتری داشته باشند.

دوماً: در حالتی که کاری که باید انجام شود از قبل کاملاً مشخص نباشد براحتی می‌توان با الگوریتم ارتباط برقرار کرد. چون تقسیم منصفانه یک کمیت ناشناخته دشوار است، از اینرو ارتباط داشتن با الگوریتم می‌تواند مفید باشد.

۵- شاخه و قید (Branch and Bound):

الگوریتم B&B با ٤ قانون مشخص می‌شود:

- قانون Branching: چگونه یک مسئله را به زیر مسئله‌هایی تقسیم کنیم.
- قانون Bounding: چگونه یک حد پایین از حل بهینه برای یک زیر مسئله را محاسبه کنیم.
- قانون Selection: کدام زیرمسئله برای انشعاب بعدی باید انتخاب شود.
- قانون Elimination: چگونه زیر مسئله‌هایی که به حل بهینه برای مسئله اولیه (Origina) نمی‌انجامند را حذف و سازماندهی کنیم.

اگر P_o مسئله می‌نیممی باشد که باید حل شود. مسیری که P_o مرتباً توسط قانون انشعاب به زیر مسئله‌های کوچکتر تجزیه می‌شود توسط یک درخت ریشه‌دار محدود $B=(P,A)$ که P مجموعه نودها است و A مجموعه یالها است، نشان داده می‌شود. به این درخت، Search tree (درخت جستجو) گویند. ارتباط یک به یک بین نودهای درخت جستجو و

زیر مسئله‌های تولید شده از تجزیه وجود دارد. ریشه این درخت P_o است. اگر زیر مسئله P_j از طریق تجزیه از زیر مسئله P_i ایجاد شود آنگاه $(P_i, P_j) \in A$ است.

اگر f(p) حل بهینه زیر مسئله P باشد و زیر مسئله P به P_1 و .... P_k تجزیه شود:

$$f(P) = \min_{j=1,\ldots,k} \{f(P_j)\} \qquad (۵-۱)$$

اگر g(P) یک حد پایین برای زیر مسئله P باشد که توسط قانون Bounding محاسبه شده است، باشد و T مجموعه زیر مسئله‌هایی باشد که بدون تجزیه می‌توانند حل شوند (به عبارت دیگر، برگهای درخت) آنگاه ویژگیهای زیر برای تابع حد پایین g متصور است:

$$g(P_i) \le f(P_i) \quad \text{for} \quad P_i \in P \qquad (۵-۲)$$

$$g(P_i) = f(P_i) \quad \text{for} \quad P_i \in T \qquad (۵-۳)$$

$$g(P_i) \le g(P_j) \quad \text{for} \quad (P_i, P_j) \in A \qquad (۵-٤)$$

همانطور که مشاهده می‌شود g یک تخمینی از حد پایین برای f است و زمانیکه P_i را بدون تجزیه بتوان حل کرد یعنی P_i برگ درخت جستجو باشد، آنگاه g دقیقاً همان حل بهینه است.

مفهوم Heuristic search (جستجوی اختیاری) یک قالب کاری را فراهم می‌کند که بتوانیم انواع قوانین انتخاب را با هم مقایسه کنیم مثل depthfirst، breadth first، best bound و ... . در یک جستجوی اختیاری تابع دلخواه h(hevristic function) بر روی

مجموعه زیر مسئله‌ها تعریف می‌شود. این تابع ترتیب انشعاب زیر مسئله‌ها را مشخص می‌کند. الگوریتم همیشه زیر مسئله با کمترین مقدار را برای انشعاب انتخاب می‌کند.

قانون حذف شامل سر تست برای حذف زیر مسئله‌ها است:

– feasibility test (بررسی امکان‌پذیری): یک زیر مسئله زمانی حذف می‌شود که بتوان ثابت کرد حل ممکن ندارد.

– lower bound test (بررسی حد پایین): یک زیر مسئله زمانی می‌تواند حذف شود که حد پایین آن بزرگتر یا مساوی با یک مقدار حل ممکن شناخته شده باشد. اگر L نشان دهنده تست حد پایین باشد، نماد P_iLP_j یعنی زیر مسئله P_i از طریق تست حد پایین زیر مسئله P_j را حذف می‌کند. یعنی P_i یک حل ممکن است و $f(P_i) \le g(P_j)$.

– dominance test (بررسی تسلط): یک زیر مسئله که تحت نفوذ زیر مسئله دیگر است باید حذف شود. زیر مسئله P_i زیر مسئله P_j را تحت سلطه خود قرار می‌دهد اگر $f(P_i) \le g(P_j)$. اگر D نشان دهنده تست تسلط باشد، آنگاه P_jDP_i برقرار باشد.

به زیر مسئله‌ای Currently dominating subproblem گویند اگر تولید شده باشد ولی تا کنون تحت سلطه زیر مسئله‌ای قرار نگرفته باشد.

حال یک اجرای خطی از لگوریتم B&B را توضیح می‌دهیم:

Active subproblem: زیر مسئله‌ای است که تولید شده ولی تا کنون نه انشعاب پیدا کرده و نه حذف شده است. زیر مسئله‌ای که در حال حاضر از آن انشعاب گرفته می‌شود یک زیر مسئله فعال است:

Active set: در هر مرحله از محاسبه مجموعه‌ای بنام مجموعه‌ای فعال وجود دارد که شامل تمام زیر مسئله‌های فعالی است که تا این لحظه از آنها انشعابی گرفته نشده است.

یک حلقه اصلی وجود دارد که گامهای زیر را مکرراً اجرا می‌کند. با کمک از قانون انتخاب یکی از زیر مسئله‌ها از مجموعه فعال برای انشعاب انتخاب می‌شود. این زیرمسئله استخراج می‌شود و با کمک قانون انشعاب به زیر مسئله‌های کوچکتری تجزیه می‌شود. برای هر یک از زیر مسئله‌های تولید شده یک حد پایین محاسبه می‌شود. اگر در حین محاسبه این حد آن زیر مسئله حل نشود، به مجموعه فعال اضافه می‌شود و قانون حذف برای هرس کردن این مجموعه بکار می‌رود. اما اگر در حین محاسبه bound، زیر مسئله حل شود، یعنی یک حل بهینه برای زیر مسئله پیدا شده است.

مقدار بهترین حل شناخته شده update می‌شود. این مقدار یک حد بالا (Upperbound) برای مقدار حل بهینه مسئله اصلی است. محاسبات تا زمانی ادامه می‌یابد که هیچ زیر مسئله فعالی وجود نداشته باشد. کار انجام شده در حین اجرا توس درخت جستجوی تولید شده نشان داده می‌شود.

تعریف Knowledge:

در حین اجرای الگوریتم B&Bدانش (Knowledge) درباره نمونه مسئله مورد حل همچنان تولید و جمع‌آوری می‌شود. این دانش از همه زیر مسئله‌های تولید شده، انشعاب داده شده هنوز تحت سلطه قرار نگرفته و حذف شده، حدود پایین برای حل بهینه، حلهای ممکن و حدود بالای بدست آمده تشکیل شده است.

تصمیم‌گیری برای آنکه در مرحله بعد چه عملی انجام شود. بعنوان مثال انتخاب زیر مسئله بعدی برای انشعاب و یا حذف کل زیر مسئله، همگی بر اساس این دانش انجام می‌شود.

Redundant knowledge (دانش افزونه) دانشی است که در گذشته توسط الگوریتم تولید شده ولی از الان به بعد هرگز مورد استفاده الگوریتم نخواهد بود. مانند حل ممکن جدیدی که منجر به باطل شدن حل ممکن قبلی می‌شود. از آنجائیکه دانش افزونه هرگز مورد استفاده مجدد الگوریتم قرار نمی‌گیرد، نیازی به نگهداری آن نیست. بر اساس ویژگیهای کامل یک الگوریتم b&b ثابت می‌شود بخشی از دانش تولید شده، افزونه است.

Valid Knowledge (دانش معتبر) دانشی است که الگوریتم هنوز نتوانسته تصمیم بگیرد که این دانش افزونه است.

٦- الگوریتم شاخه و قید موازی: (Parallel B&B Algorithms):

عده‌ای الگوریتمها را در دو سطح low و high دسته‌بندی می‌کنند و عده‌ای الگوریتمها را بصورت سنکرون و آسنکرون تقسیم‌بندی می‌کنند. (low معادل با سنکرون و high معادل با آسنکرون) الگوریتمها در دو سطح lowو high می‌توانند موازی شوند:

موازی سازی در سطح low: الگوریتم خطی بعنوان نقطه شروع داده می‌شود و فقط بخشی از الگوریتم موازی می‌شود، بطوریکه محاوره بین بخش موازی و دیگر بخشهای الگوریتم تغییر نمی‌کند. در الگوریتمهای B&B، «محاسبه حد پایین»، «انتخاب زیر مسئله

برای انشعاب بعدی» یا «کاربرد قانون حذف» می‌تواند توسط چندین پروسس بطور موازی انجام شود.

از آنجائیکه محاوره بین بخشهای مختلف الگوریتم تغییر نمی کند، موازی سازی در سطح پایین در کل الگوریتم تاثیری ندارد. در کل الگوریتم موازی تولید شده، رفتار الگوریتم خطی اصلی را بازسازی می‌کند. (یعنی در الگوریتم B&B از همان زیر مسئله‌ها و به همان ترتیب انشعاب خواهد گرفت).

بنابراین نیاز به مطالعه کل الگوریتم موازی نیست بلکه فقط کافیست آن بخش که واقعاً موازی است را مطالعه کنیم. یکبار که اثر موازی سازی شناخته شود، رفتار الگوریتم موازی کاملاً قابل پیشگویی خواهد بود.

موازی سازی در سطح high: آثار و نتایج این نوع موازی سازی محدود به یک بخش از الگوریتم نیست بلکه در کل الگوریتم اثر می‌گذارد. این الگوریتم موازی اساساً متفاوت است. و کار انجام شده در این نوع الگوریتم موازی الزاماً با کار انجام شده در الگوریتم خطی یکسان نیست.

ترتیب کار انجام شده هم می‌تواند متفاوت باشد و نیز ممکن است بخشی از کار انجام شده توسط الگوریتم موازی اصلاً توسط الگوریتم خطی انجام نشده باشد و یا برعکس.

در الگوریتمهای B&B چند تا تکرار از حلقه اصلی می‌تواند بصورت موازی انجام شود. چون تکرارهای این حلقه نسبتاً مستقل از یکدیگر هستند، ترتیبی که این تکرارها مجدداً مرتب می‌شوند یا انجام چندین تکرار بطور موازی در صحت الگوریتم تاثیری ندارد.

در ادامه ما الگوریتمهای B&B موازی را در سطح high (یعنی چندین تکرار در حلقه اصلی را بطور موازی انجام می‌دهیم) مطالعه می‌کنیم.

راههای مختلفی برای توضیح موازی سازی در الگوریتمهای B&B وجود دارد، دوباره بین موازی‌سازی سنکرون و آسنکرون تفاوت قائل می‌شویم. در حالت سنکرون اجرای حلقه اصلی الگوریتم B&B به subtask هایی تقسیم می‌شود که هر کدام بصورت خطی اجرا می‌شوند، اگرچه در اجرای هر subtask ی ممکن است موازی‌سازی صورت گیرد. اما در حالت آسنکرون، اجرا به اینصورت تقسیم نمی‌شود، بلکه subtask ها مرتبط به یک یا چند تکرار از حلقه اصلی هستند و یا برعکس، یک تکرار از حلقه اصلی می‌تواند شامل چندین subtask باشد.

الگوریتم موازی شاخه و قید سنکرون :

هر بخش از الگوریتم که گامهای زیادی برای اجرا داشته باشد یک کاندید طبیعی برای موازی سازی است. کاندیداهای ممکن عبارتند از:

۱- lower bound calculation (محاسبه حد پایین): بعضی از الگوریتمهای B&B از تابعهای حد پایینی استفاده می‌کنند که محاسبه آنها دشوار است. وابستگی به یک تابع حد پایین استفاده شده خاص موازی سازی در این بخش از الگوریتم را نشان می‌دهد. اگرچه الگوریتمهای محاسبه حد پایین بطور ذاتی خطی هستند مثل الگوریتمهای greedy.

۲- Selection (انتخاب): در بعضی از مراحل اجرا تعداد زیر مسئله‌ها در مجموعه فعال ممکن است خیلی زیاد باشد، لذا انتخاب زیر مسئله بعدی برای انشعاب و استخراج آن زیر مسئله مستلزم مقدار کار زیادی است که این کار می‌تواند بطور موازی انجام شود.

۳- Elimination (حذف): بررسی اینکه bound یک زیر مسئله همچنان بهتر از حل ممکن شناخته شده تا کنون است یا نه ساده است. با اینحال بکار بردن dominance test و بررسی اینکه یک زیر مسئله می‌تواند یک حل ممکن تولید کند خیلی دشوار است. زیرا dominance test مستلزم مقایسه زیر مسئله فعلی با تمام زیر مسئله‌هایی که تولید شده‌اند ولی تا کنون تحت سلطه قرار نگرفته‌اند می‌باشد. حال این امکان وجود دارد که (بخشی از) این تستها بطور موازی انجام شوند.

٤- Branching (انشعاب) : بجای انتخاب و انشعاب یک زیر مسئله در یک زمان می‌توان چندین زیرمسئله را یکبار انتخاب کرد و بطور موازی آنها را انشعاب داد. برای آنکه عنصر پردازشی کارایی بیشتری داشته باشد، تعداد زیر مسئله‌های فعال باید حداقل برابر با تعداد عناصر پردازشی باشد.

سطح موازی سازی که از طریق روش ٤ تولید می‌شود بالاتر از سطح موازی سازی تولید شده توسط ۱ و ۲و ۳ است. یعنی روش ٤ موازی سازی در سطح high است.

کاری که در حین انشعاب از چندین زیرمسئله بطور موازی انجام می‌شود می‌تواند متفاوت از کاری باشد که در اجرای الگوریتم خطی مشابه انجام می‌شود، بعبارت دیگر درخت جستجوی تولید شده می‌تواند متفاوت باشد، چون بعضی از زیرمسئله‌ها که بطور

موازی انشعاب یافته‌اند ممکن است در حالت خطی اصلاً حذف می‌شوند یا هرگز تولید نمی‌شوند. انشعاب چندین زیر مسئله بطور موازی اساساً منجر به تغییر استراتژی جستجو می‌شود. به دلیل اینکه چندین subproblem بطور موازی در حال انشعاب دادن هستند. دانش بدست آمده در یک نقطه از اجرا متفاوت از دانش بدست آمده در حالت خطی است، لذا این مسئله می‌تواند بر روی انتخاب زیر مسئله‌ها برای انشعاب بعدی تاثیر بگذارد. نتایج این تغییر کاملاً واضح نیست، اما انواع آنومالیها رخ می‌دهد و تسریع‌ها و کاهش‌های بی‌دلیل ایجاد می‌شود.

نتیجه الگوریتم موازی سنکرون قطبی است. به جهت این همزمانی، دانش بدست آمده همواره از روش یکسانی Combine می‌شود. بنابراین اجراهای متوالی از الگوریتم همواره منجر به جواب مشابهی می‌شود.

سیستم‌های SIMD برای اجرای الگوریتمهای B&B موازی سنکرون مناسب نیستند.

– الگوریتم موازی شاخه و قید آسنکرون:

موازی‌سازی آسنکرون را فقط از طریق موازی کردن الگوریتم B&B بطور کامل می‌توان توضیح داد. از آنجائیکه تکرارهای حلقه اصلی الگوریتم B&B نسبتاً مستقل از یکدیگر هستند، لذا سازماندهی مجدد این تکرارهای انجام شده تاثیری بر صحت الگوریتم ندارند. برای تمام زیرمسئله‌های فعال تا نقطه‌ای از زمان، این مسئله برقرار است که، آنها نه انشعاب پیدا کردند و نه حذف شده‌اند، بنابراین پیش شرطهای حاکم بر این زیر مسئله‌ها

معتبر باقی می‌ماند. اگرچه سازماندهی مجدد دسته‌بندی ممکن است منجر به تولید درخت جستجوی متفاوتی شود.

ایده اصلی در موازی سازی آسنکرون، این است که چندین تکرار از حلقه اصلی بطور موازی انجام شود. هر پروسس مجموعه تکراری را که به او اختصاص یافته انجام می‌دهد. به محض آنکه پروسسی تکرارهایش را تمام کرد، اجرای مجموعه جدیدی از تکرارها را بدون آنکه منتظر اتمام کار دیگر پروسسها شود، آغاز می‌کند.

۷- پارامترهای الگوریتمهای شاخه و قید موازی آسنکرون:

ابتدا به تعاریف زیر می‌پردازیم:

Knowledgebase: موجودیتی که شامل دانش است. دانش تولید شده توسط پروسسهای مختلف به این موجودیت منتقل می‌شود و یک پروسس از این طریق به دانش مورد نظر دست می‌یابد.

Sharing the Knowledge: انتقال دانش تولید شده به Knowledge base

Using the Knowledge: دسترسی به Knowledge base و نتیجتاً استفاده از دانش

Knowledge hand ling: شامل پروسه کامل انتقال دانش و دسترسی به دانش است.

دانش ذخیره شده در Knowledge base در دو موقع مختلف استفاده می‌شود. اولاً، وقتی یک پروسس بخواهد تصمیمی بگیرد، باید به Knowledge baseها دسترسی پیدا کند و تصمیم را بر پایه این دانش بگیرد. دوماً، وقتی که Knowledgebwe ی update می‌شود، بعبارت دیگر وقتی دانشی تولید می‌شود. یک پروسس می‌تواند به این تغییر عکس‌العمل

نشان دهد. از این رو استراتژی برای دستیابی به knowledge base ها، عکس‌العمل به تغییرات آنها باید مشخص باشد.

امکان آنکه درک درستی از کار لازم برای اجرای الگوریتم B&B بدست آوریم بدون اجرای واقعی آن وجود ندارد. بنابراین تنها راه ممکن برای تقسیم عادلانه کار بین پروسسهای مختلف تقسیم کار به محض تولید شدن یا به عبارت دیگر بصورت داینامیک در حین اجرا است.

برای تقسیم کار باید یک واحد پایه برای کار (Units of work) انتخاب شود. یک مثال از واحد کاری، انشعاب از یک زیر مسئله می‌تواند باشد. واحدهای کاری که منتظر تکمیل شدن هستند، دانشی را درباره مسئله مورد حل تشکیل می‌دهند و در Knowledge base ذخیره می‌شوند. هر زمان که پروسسی بی‌کار می‌شود، به این Knowledge base ها مراجعه می‌کند و کار جدیدی را دریافت می‌کنند، یعنی واحدکار جدید از یک Knowledge base استخراج شده و جهت اجرا به پروسس داده می‌شود. سرانجام باید مشخص شود که وقتی پروسسی واحد کاری را که بر روی آن کار انجام می‌داد به اتمام رساند، چه اتفاقی می‌افتد. دو حالت وجود دارد: قبل از شروع کردن واحد کار بعدی، پروسس می‌تواند منتظر پروسسهای دیگر برای اتمام واحد کاریشان شود و یا پروسس می‌تواند کار بر روی واحد کار جدید را بلافاصله و بدون انتظار آغاز کند.

در زیر پارامترهای مهم در الگوریتمهای B&B موازی را شرح می‌دهیم:

۱-۷- Knowledge sharing:

knowledge base ها بر اساس پروسسی که به آنها دسترسی دارد و دانشی که در آنها ذخیره می‌شود دسته بندی می‌شوند، بنابراین دسته‌بندیهای زیر را خواهیم داشت:

- Knowledge base , global knowledge base ی که در دسترس همه پروسسها است.
- Knowledge base , local knowledge base فقط در دسترس زیر مجموعه‌ای از پروسسها است.
- knowledge base, global knowledge base که فقط در دسترس یک پروسس است اما توسط چندین پروسس می‌توانند update شود.
- knowledge base, Compelete knowledge base ی که همه دانش تولید شده توسط تمام پروسسها، به آن منتقل شده است.
- knowledge base, partial knowledge base ی که فقط بخشی از دانش تولید شده به آن منتقل شده است.

مثلاً الگوریتم B&B موازی می‌تواند از یک Partial knowledge base استفاده کند که شامل فقط بخشی از زیر مسئله‌های فعال است.

خاصیت global compelete knowledge base ها این است که تخمین خوبی از دانش تولید شده تا کنون را می‌دهند، اما عیب آن ایجاد کردن گلوگاه (Bottleneck) است چون

در یک زمان بر روی Knowledge base فقط یک پروسس می‌تواند عملیات انجام دهد. Local, partial knowledge base مزیتش این است که از ایجاد گلوگاه جلوگیری کرده اما برخی از پروسسها ممکن است بر روی واحدهای کاری نامناسبی کار کنند. (مثلاً انشعاب از زیر مسئله‌هایی با مقدار bound بالا) چون این دانش جزئی است و اطلاعات کامل را نمی‌دهد پس تخمین خوبی برای انتخاب واحد کاری مناسب و یا هرس کردن واحدهای کاریی که منتظر تکمیل شده هستند، نمی‌باشد.

چندین local complete knowledge base مزیتی که دارد این است که هر knowledge baseی تخمین خوبی از دانش بدست آمده تا کنون را می‌دهد و گلوگاه ایجاد شده در اثر دسترسی پروسسهای خیلی زیاد در مدت زمان بسیار کوتاهی کاهش پیدا می‌کند. با این حال برای حفظ سازگاری محتویات آنها رعایت Mutually Exclusive access به دانش خاص دشوار است.

مزیت global partialknowledge base کاهش تعداد عملیاتی است که پروسسهای مختلف انجام می‌دهند (دانش کمتری توسط الگوریتم استفاده می‌شود.) عیب آنهم این است که دانش معتبر درباره مسئله بطور عمدی از طرف الگوریتم دور انداخته می‌شود بهترین knowledge sharing زمانی بدست می‌آید که هر پروسس تمام دانشی را که تولید کرده بلافاصله به تمام knowledge base های وابسته منتقل کند. بعضی از sharingها، اگرچه پیچیدگی ارتباط را افزایش می‌دهند. اما یک trade offی بین تعداد واحدهای کاری حذف شده و میزان Communication انجام شده وجود دارد.

۲-۷- Knowledge use:

یک پروسس در دو موقعیت دانش را استخراج می‌کند:

۱- وقتی پروسس بخواهد تصمیم بگیرد.

۲- وقتی دانش جدید تولید شده توسط دیگر پروسسها در دسترس قرار گیرد.

وقتی پروسسی می‌خواهد تصمیمی بگیرد، به knowledge base جهت گرفتن دانش مورد نظر دسترسی پیدا می‌کند تا تصمیم را بر اساس آن دانش اتخاذ کند. استراتژی دستیابی (Access strategy) مشخص می‌کند که به کدام knowledge base و چه زمانی مراجعه کند.

برای آنکه بتوان از دانش جدیداً تولید شده توسط دیگر پروسسها استفاده کرد، پروسس باید بتواند ابتدا knowledge base های update شده را شناسایی کند. و به دانش جدید واکنش نشان دهد. یعنی از وجود آن آگاه شود و آنرا در نظر گیرد و بر اساس ان تصمیم گیرد که چگونه ادامه دهد.

استراتژی واکنش (reaction strategy) مشخص می‌کند که یک پروسس چگونه از تغییر یک knowledge base آگاه شود، و چگونه به این تغییر واکنش نشان دهد.

دو راه اساسی برای آگاهی از تغییر knowledge base وجود دارد:

- یک پروسس می تواند knowledge base را سرکشی کند، یعنی بطور مرتب بررسی کند که آیا تغییر کرده است.

- یا پروسس می‌تواند هر زمان که knowledge base تغییر کرد اینتراپت دهد، یعنی هر زمان knowledge base تغییر کرد، پروسس متوجه آن شود و فعالیت جاریش را متوقف کند و روتین اینتراپت را آغاز کند و بعد ازاتمام روتین اینتراپت دوباره فعالیت سابق خود را از سر خواهد گرفت.

۳-۷- Dividing the work:

Units of work را می‌توان به دلخواه انتخاب کرد. البته در عمل واحد کاری با Communication complexity ارتباط متقابل دارد. اگر واحدها خیلی کوچک باشند، شبکه ارتباطی اشباع می‌شود. در الگوریتم B&B موازی می‌توان از چندین واحد کاری بطور همزمان استفاده کرد.

در اصل واحدهای کاری که منتظر کامل شدن هستند، دانشی درباره مسئله مورد حل تشکیل می‌دهند. از این‌رو تقسیم کاربین پروسسها از طریق knowledge base ها انجام می‌شود. کار جدید تولید شده توسط پروسسها به knowledge baseها منتقل می‌شود.

Active unit of work واحدکاری است که توسط پروسس تولید شده و اجرای آن آغاز شده اما هنوز کامل نشده است.

همانند الگوریتم B&B باید با یک قانون انتخاب، واحدکاری بعدی را انتخاب کنیم تا کار بر روی آن آغاز شود و نیز تابع دلخواهی مورد استفاده قرار می‌گیرد که بر اساس آن تابع واحد کاریی انتخاب می‌شود که کمترین مقدار را داشته باشد.

knowledge baseی را که شامل active unito of work ها نباشد dried up گویند. برای جلوگیری از dry up شدن از استراتژیهایی بنام load balancing strategy استفاده می‌کنیم که این استراتژیها نمی‌گذارند knowledge base ها شامل واحدهای کاری غیر جذاب باشند البته برای این کار باید دانش درباره واحدهای کار active را بدون تکرار در Portial knowledge base های مجزایی ذخیره کنیم.

۷-٤- Synchronicity :

پارامتر آخر در الگوریتمهای B&B موازی high level (آسنکرون) این است که ببینیم وقتی یک پروسس واحدکاری را که بر روی آن عملیات انجام می‌داد تمام کرد چه اتفاقی می‌افتد. دو حالت وجود دارد:

- قبل از دسترسی به knowledge base و شروع واحد کاری جدید، پروسس می‌تواند صبر کند تا دیگر پروسسها عملیاتشان را کامل کنند.
- پروسس می‌تواند به knowledge base رجوع کند و کاربر روی واحد کاری جدید را بلافاصله و بدون انتظار آغاز کند.

همزمانی کامل به این معنی نیست که در هر نقطه از زمان پروسسها دستورالعمل یکسانی را اجرا می‌کنند. در مسئله همزمانی پروسسها، اگر task هایی که باید انجام شوند سایز یکسانی ندارند و یا پروسسها توان یکسانی ندارند، زمانیکه پروسسها منتظر یکدیگر می‌شوند تا task هایشان را انجام دهند، قدرت محاسبه از دست خواهد رفت. علاوه بر

این، همزمانی مستلزم communication است چون پروسسها باید بدانند که پروسسهای دیگر آیا taskهایشان را کامل کرده‌اند یا خیر ؟!

مزیت همزمانی این است که با تقسیم کاربین پروسسهای مختلف و انتقال دانش تولید شده به knowledge base های مختلف تلاش می‌کنیم تا زمانیکه این اعمال انجام می‌شود و در نقطه همزمانی دانش stable باشد یعنی در خلال انجام این کارها دانش جدیدی تولید نمی‌شود.

خصوصیت پروسسهای آسنکرون دقیقاً برعکس است. همزمانی پروسسها منجر به قطعی شدن نتیجه می‌شود. اگر همه پروسسها کاملاً همزمان نباشند، انتقال دانش تولید شده به knowledge baseها و آگاهی از دانش جدید و سپس استفاده از آن باعث ایجاد رفتار غیرقطعی از الگوریتم می‌شود. این اختلاف بدلیل فاکتورهای محیطی است که تحت کنترل الگوریتم نمی‌باشد. مثلاً ممکن است دیگر پروسسهای سیستم موجب تصادم عناصر پردازشی متصل به شبکه شوند، و یا clock های عناصر پردازشی با هم اختلاف پیدا کنند.

اگر knowledge baseی که توسط یک پروسس update می‌شود، اجازه دهیم که پروسس دیگر به آن دسترسی داشته باشد، آنگاه الگوریتم مسیر متفاوتی را طی می‌کند و در نتیجه حل متفاوتی خواهیم داشت. حتی اگر دو اجرای پی‌در پی از الگوریتم نتیجه یکسانی داشته باشند، کار انجام شده برای اجرای آنها ممکن است کاملاً متفاوت باشد. به این رفتار خاص fluctvation anomaly گویند.

رفتار غیر قطعی اثری بر صحت الگوریتم B&B موازی ندارد، غیر قطعی بودن خاصیت زیر مسئله‌های موجود در activeset را تغییر نمی‌دهد.

۸- پیچیدگی و تسریع (Complexity & Speedup):

حال می‌خواهیم نتیجه استفاده از موازی‌سازی را در زمان مورد نیاز برای اجرای الگوریتم بدانیم و آنرا اندازه بگیریم. در الگوریتمهای موازی سنکرون این آثار با نماد Speed up در efficiency توصیف می‌شوند، بطوریکه بصورت تابعی از تعداد عناصر پردازشی مورد استفاده تغییر می‌کنند.

Speedup (تسریع) معیاری از کاهش کل زمان اجرا است که به جهت موازی سازی صورت گرفته و به گونه زیر تعریف می‌شود:

(۸-۵)

$$\text{تسریع} = \frac{\text{تعداد دستورالعملهای لازم برای اجرای بهترین الگوریتم خطی توسط یک عنصر پردازشی}}{\text{تعداد دستورالعملهای لازم برای اجرای الگوریتم}}$$

Efficiency (کارایی) اطلاعاتی را درباره این تسریع در مقایسه با تسریع ایده‌آل به ما می‌دهد و به صورت زیر تعریف می‌شود:

(۸-۶)

$$\text{کارایی} = \frac{\text{تسریع}}{\text{تعداد عناصر پردازشی استفاده شده}}$$

اما نمادهای تسریع و کارایی بالا قابلیت توصیف اجرای الگوریتم موازی آسنکرون را بطور مناسب ندارند. اولاً: آنها زمان لازم برای سنکرون کردن را در نظر نمی‌گیرند. دوماً: فرض می‌کنند زمان لازم برای یک ارتباط ثابت است. در حالیکه این زمان نه تنها وابسته

به تعداد ارتباطات و هماهنگی‌ها است بلکه به سیستم کامپیوتر موازی خاصی که الگوریتم بر روی آن اجرا می‌شود هم بستگی دارد. سوماً: فرض می‌کنند که تمام عناصر پردازش استفاده شده یکسان هستند، در حالیکه در اکثر سیستمهای کامپیوتری موازی، خصوصاً سیستمهای loosely coupled عناصر پردازشی متفاوتی وجود دارد. برای یک الگوریتم موازی آسنکرون این اختلاف در توان پردازشی اهمیتی ندارد چون زمان لازم برای اجرای صورت زمانی است که ضعیف‌ترین عنصر پردازشی لازم دارد.

در الگوریتمهای آسنکرون این امکان وجود دارد که با توزیع مجدد کاربین عناصر پردازش بتوانیم زمان اجرا را کوتاهتر کنیم.

برای توصیف اجرای الگوریتمهای موازی آسنکرون نمادهای توضیح داده شده بالا بایستی عمومیت پیدا کنند. بنابراین نمادهای perfect parallel time و achieved efficiency را معرفی می‌کنیم.

موازی سازی «کامل» است اگر:

استفاده از موازی‌سازی بر روی کاری که باید انجام شود اثری نگذارد و کار به ۲ طریق زیر قابل تقسیم بین عناصر پردازشی باشد.

۱- هر عناصر پردازشی به مقدار زمان یکسانی برای انجام کار مشترک نیاز داشته باشند.

۲- تمام شرکا از یکدیگر مستقل باشند.

بنابراین عناصر قدرتمندتر سهم بیشتری از کار را بعهده می‌گیرند، در حالیکه عناصر ضعیف‌تر سهم کمتری دارند. با توجه به فرض اول یعنی محاوره بین عناصر پردازشی

زمانی را صرف نمی‌کند. Perfect parallel time زمان لازم برای اجرای موازی کامل یک الگوریتم است.

اگر W تعداد دستورالعملهای انجام شده توسط الگوریتم خطی مشابه باشد و S_i سرعت عنصر پردازشی iام بطوریکه I=1,... , N (یعنی تعداد دستورالعملهایی که عنصر پردازشی iام در واحد زمان می‌تواند انجام دهد.)

(الگوریتم خطی مشابه را به صورت الگوریتم موازی که توسط یک پروسس اجرا می‌شود تعریف می‌کنیم.)

$$PPT = \frac{W}{\sum_{i=1}^{N} S_i} \qquad (۸-۷)$$

Achieved efficiency خارج قسمت PPT بر زمان واقعی مورد نیاز برای اجرای موازی است. همانطور که از این تعریف می‌بینیم، زمان لازم برای ارتباط و سنکرون کردن را هم در نظر می‌گیرد.

از آنجائیکه فرض بالا در مورد تقسیم‌پذیری، استقلال‌پذیری و محاوره در عمل غیرواقعی است، لذا perfect parallel time یک حد پایین از زمان لازم برای اجرای موازی روی یک سیستم کامپیوتر موازی است.

$$PPT \le T_n(I) < T_1(I) \qquad (۸-۸)$$

PPT: زمان اجرای موازی کامل

$T_n(I)$: زمان لازم برای حل نمونه مسئله I توسط الگوریتم موازی با n عنصر پردازشی

$T_1(I)$: زمان لازم برای حل نمونه مسئله I توسط الگوریتم خطی مشابه توسط یک عنصر پردازشی

۹- مثالی از یک الگوریتم شاخه و قید موازی آسنکرون (Boulder Alglorithm):

حال می‌خواهیم چندین زیر مسئله را با کمک از دانش تولید شده بطور موازی و بهینه‌ترین راه انشعاب دهیم. منظور از «بهینه» یعنی در اجرای الگوریتم B&B موازی فقط آن زیر مسئله‌هایی را انشعاب می‌دهیم که در الگوریتم خطی مشابه انشعاب داده می‌شدند. انشعاب دادن زیر مسئله‌هایی که حذف شده‌اند یا هرگز در الگوریتم خطی مشابه تولید نشده‌اند. به نظر می‌رسد که کار بیهوده‌ای باشد که بطور غیر ضروری اجرا را کاهش می‌دهد. ما الگوریتم B&B موازی را که از یک global knowledge base برای تقسیم کار بین پروسسها استفاده می‌کند، توسعه دادیم. Basic unit of work آن انشعاب از یک زیر مسئله منفرد است که شامل محاسبه حدود پایین برای زیر مسئله‌های تولید شده نیز می‌باشد. از آنجائیکه یک واحد کار، انشعاب از یک زیر مسئله است، بنابراین hevristic function تعریف شده بر روی واحد کار مشابه hevristic function تعریف شده روی زیر مسئله‌ها می‌باشد.

الگوریتم آسنکرون را انتخاب کردیم چون می‌خواهیم الگوریتم را روی سیستم loosely coupled اجرا کنیم که ارتباطات خیلی کند و آهسته است. لذا نمی‌خواهیم زمان صرف همزمانی ارتباطات شود.

امکان استفاده از دانش بدست آمده تا کنون به بهترین شکل باعث افزایش پیچیدگی ارتباطی الگوریتم می‌شود. اگر چه تابع حد پایین مورد استفاده مستلزم ارتباطات زیاد است. علاقه‌مندیم که این افزایش در پیچیدگی ارتباط روی کاهش زمان محاسبه تاثیر بگذارد. الگوریتم شامل یک پروسس اصلی (master) و چندین پروسس فرعی (Slave) می‌باشد، بطوریکه پروسس master تمام تصمیمات مهم را اتخاذ می‌کند و پروسسهای slave به گونه‌ای که master به انها دستور داده رفتار می‌کنند. پروسس master مجموعه فعال را نگه می‌دارد و تصمیم می‌گیرد که کدام یک از زیر مسئله‌ها باید انشعاب پیدا کنند و چه زمانی؟! پروسسهای Slave، انشعاب واقعی و محاسبه حدود پایین برای زیر مسئله‌های تولید شده را انجام می‌دهند. (به این الگوریتم Boulder گویند).

برای انکه همه چیز را به طور صحیح مدیریت کنیم. پروسس master همه دانشی که تا الان توسط slave ها تولید شده را جمع‌آوری می‌کند (مثل زیر مسئله‌های تولید شده، حلهای ممکن پیدا شده و ...) هر زمان که master از بی‌کاری یک slave مطلع شود. زیر مسئله‌ای با کوچکترین مقدار انتخابی (heuristic value) از مجموعه فعال استخراج می‌کند و آنرا برای انشعاب به slave می‌دهد. اگر هیچ زیر مسئله فعالی وجود نداشته باشد، پروسس slave بطور قومتی در صفی از slaveهای بی کار قرار می‌گیرد تا زمانیکه زیر مسئله‌های جدید در دسترس قرار گیرند. اجرای الگوریتم B&B زمانی خاتمه می‌یابد که تمام slaveها در صف بی‌کار باشند.

یک slave زیر مسئله دریافت شده از master را انشعاب می‌دهد. سپس زیر مسئله‌های تولید شده به master فرستاده می‌شود.

این باعث می‌شود که master از این دانش به محض امکان بتواند استفاده کند. در نتیجه در ابتدای اجرا تمام slaveها می‌توانند فعال باشند. بعد از کامل شدن عمل انشعاب، slave با آگاه کردن master از اینکه بی‌کار شده است درخواست کار جدیدی می‌کند.

در نگاه اول عجیب است که یک slave همه زیر مسئله‌های تولید شده را به master بفرستد و سپس بجای آنکه یکی از آنها را که خودش تولید کرده انشعاب دهد، از master سئوال کند.

اما علت این است که یک slave دانش کافی درباره کل مسئله ندارد. البته زیر مسئله تولید شده تنها انتخاب خوب هم نخواهد بود، لذا با پرسیدن slave از master درباره کار جدید، مقدار کارغیر ضروری انجام شده حداقل می‌شود.

۱-۹- پیاده سازی الگوریتم:

برای اجرای الگوریتم نیاز به سیستمی داریم که بتواند پیامی از یک پروسس به پروسس دیگر بدون همزمانی پروسسها بفرستد و توانایی بافرکردن پیامها را داشته باشد. همچنین پروسس باید توانایی ادامه محاسباتش را بلافاصله بعد از ارسال پیام داشته باشد بدون آنکه منتظر دریافت پیام شود.

این نوع ارتباط بسیار مناسب است چون هر زمان که یک slaveی دانش جدیدی بدست می‌آورد باید این دانش را به master بفرستد. بلافاصله بعد از آن slave به ادامه عملیاتش می‌پردازد و نیازی نیست که منتظر شود تا master پیام را دریافت کند. سیستم باید پیام را بافر کند چون این امکان وجود دارد که یک پروسس message دومی را قبل از آنکه پروسس دیگر پیام اول را بخواند ارسال کند. اگر سیستم ارتباط داخل پروسسها را به این شکل فراهم نکند، امکان اجرای الگوریتم وجود دارد گرچه توان محاسبه به جهت همزمانی غیر ضروری از دست می‌رود.