＜特集：炉物理研究への PC クラスタの利用・並列計算の基礎知識＞

並列計算の基礎知識

原子燃料工業(株) 巽　雅洋
tatsumi@nfi.co.jp

1. 並列計算への招待

並列計算と聞いて、皆さんはどういう印象を受けるでしょうか？　「なんだかよく分からないが、とにかく難しそう…」というのが、私がいろんな方から伺うご意見の大部分です。最近こそ聞くことは少なくなりましたが、「『並列計算機』を買えば、手元にあるコードが速くなるの？」という質問も、昔は結構あったと思います。このような誤解は、『並列計算機』を用いておこなう『並列計算』について、正しい認識を持っていただければすぐに解けると思います。本章では、まず、『並列計算』に対して正しく理解していただくことを目標に、並列計算にかかわるいくつかの項目について述べていきたいと思います。

1.1 もっと速く！

計算機[1]と向かい合っていると、「まぁ、こいつはなんと融通のきかないことか」とか「もっと速く計算してよ！」と思うことがあるでしょうか？　計算機は、ご存知のとおり、決められた計算は正確にこなしてくれます。しかし、計算のやり方は、プログラムという形で人間様[2]が指示してやる必要があります。プログラムには、「あれとこれを足して、次にこれで割る」といった手順がずらずらっと示されています。つまり計算機は、人間様が示すプログラムどおりに計算するしか能がないわけです。(指示したとおりの処理が素早く実行されたときには、「お前はいいやつだ！」とも思ったりもしますが… 。) そういう意味では、計算機は「愛しい」というか「愛らしい」存在ですが、自分が与えたプログラムの出来が悪く、計算時間も多くかかってしまうと、非が自分にあることをすっかり忘れて先ほどのようなため息が出てくるわけです。

結局のところ、必要となる計算時間は、最終的にプログラムを組む自分の腕にかかってきます。ですから、なんとかして計算時間を短くしたいと思うのが人情でしょう。人によっては、「計算機なんて、黙っていても速くなるんだから、そんなに気合いれなくてもいいんじゃないの？」と思う人もいるかもしれません。しかし、「F1 レーサーが自分の腕を頼りにマシンの能力を最大限引き出す」あるいは、「登山家がより高い山を目指す」というのと同じで、多くの人が、目の前にある計算機の最大限の能力を引き出したいと考えるのはもっとものことだと思います。

「速い計算機を使っている。アルゴリズムも最速のものだ。もうやることはない！ 」と思ったあなた、「コードをもっと速くしたい！ 職場には計算機がいくつかあって、自由に

---

[1] 本章では、EWS(エンジニアリング・ワークステーション)、パソコン等を含むコンピュータのことを「計算機」と呼ぶことにします。
[2] プログラムを組む人を指します。

つかえるんだけど…」と思ったあなた！　「解きたい問題があるけど、1 台の計算機ではメモリが足りなくて…」と困っているあなた！　『並列計算』があたなを待っています！

1.2 筆者と並列計算の出会い

ここでちょっと趣向を変えて、私事ではありますが、私が『並列計算』（まだ、括弧つきです）に関わる（のめり込む？）にいたった経緯についてお話させていただきたいと思います。

私が在籍していた大学には、情報処理教育センター（通称、情教センター）という施設があり、1991 年に NeXT 社のワークステーションが大量[3]に導入されました。これは学生に自由に開放され、筆者も情教センターに足しげく通ったものでした。NeXT ワークステーション[4]には、複数の計算機が協力してひとつの処理を行う Zilla というアプリケーションがありました。これは、「**複数の計算機を協調して動かすことによって、一つの処理をより短時間で実行する**」という**『並列計算』の概念**が視覚的に理解できる、優れたアプリケーションでした。ただ、具体的にどうやったら並列計算（もう括弧はなし）が可能となるのを理解するためには、OS である NeXTStep の詳細を理解するために大量の英語のドキュメントを読む必要があり、非常に敷居が高かったことを覚えています。

それからしばらくして、私が大学院 1 年生だった 1994 年当時、『並列計算』はまだ一部の人たちのものでした。というのも、本格的な並列計算ができる計算機は、大学や研究所等に設置されている大型機[5]しかなかったからです。そのようなマシンといえば安くても数千万円しましたので、研究室でおいそれと買えたものではありませんでした。そこで、計算機センターにあるマシンを使うということになるのですが、計算機費用の課金が結構かかるということがわかり、これまたあまり使うことができなかったのでした。

そうこうしているうちに、UNIX Magazine という雑誌に、PVM と呼ばれるソフトウェアの解説記事がありました。そこには、「ネットワークで接続された計算機を、あたかも単一の仮想計算機のようにみせる」とありました。PVM は、”Parallel Virtual Machine”の意味らしく、「なるほど」と思ったものです。その解説記事には、PVM の仕組みや実際のプ

---

[3] 阪大の「情教センター」には、約 400 台の NeXTStep マシンが導入されました。当時としては、これほど大量に導入されたのは阪大が初めてで、その視察のため CEO の Steve Jobs 氏が来訪した際には、ミーハーな筆者は Jobs をこの目で見たい！ということで講演会に行ったのでした。

[4] NeXTStep は、ある意味時代を先取りしすぎてそのメリットが理解されず、不運の星に生まれたといえるでしょう。10 年以上前のものとは思えないほどモダンな設計で、今では広く受け入れられている、オブジェクト指向、マイクロカーネル、マルチスレッド、PostScript の技術を用いた野心的な OS でした。現在は、Mac OS X として生まれ変わりつつありますが、Windows 帝国の牙城を崩すのは難しそうです。「よい物が普及するとは限らない」というのを体現している良い例でしょう。

[5] Connection Machine 社の CM-5、Cray Research 社の Cray Y/MP シリーズ、Sillicon Graphics 社(SGI)の Onyx 等。阪大の大型計算機センターにも Onyx がありました。

ログラミング方法が詳細に解説されていて、非常によく理解できたのを覚えています。「これからは PVM や！」[6]と感じた一瞬でした。それから 1 年後、運良く原子力学会の交換留学生プログラムで米国のアルゴンヌ国立研究所(ANL)にしばらく行くことができました。どうして留学先を ANL にしたかというと、その 1 年前に阪大から留学した学生がいて、その人から「ANL はええとこやでぇ」と聞いていたからでした。あと、「ANL には並列計算機があるらしい」ということが分かったので、「ANL しかない！」と直感したのでした。　そのときには、「並列計算機を使いまくって何かしたい」と漠然と考えていたのですが、具体的なことについては何も分からない状況でした。

ANL では、世話役から 1 冊の本を渡されました。タイトルは"Using MPI"となっており、その時に初めて MPI なるものがあるということを知りました。よくよく読んでみると、「MPI は PVM と同じく、並列計算を行うときに使うもの。計算機同士の通信手順を決めたもの[7]」だということが理解できました。また、「MPI は、業界統一標準を目指したもの」「MPI は効率が良い（らしい)」ということも理解できました。で、そのときになって初めて、この MPI の標準作りに ANL が中心的役割を果たしているということを理解し、どこかの CM ではありませんが、「ANL にして良かった」と自分の直感を褒めた(?)のでした。

並列計算機を使いまくりたい…という強いオーラを発していたのかどうかわかりませんが、世話役からは「まずはネットワークで接続されている計算機を使って、基礎的なことを調査してみよう」と言われました。そのときは「並列計算機を使いまくるという野望」が打ち砕かれ少し(?)ショックでしたが、後から思うとこれには二つの意味合いがあったのでしょう。一つは、ネットワークで接続されたワークステーションをひとまとめにして使ういわゆる計算機クラスタと、並列計算機の性能の差異について体感させること。もうひとつは、素人がいきなり並列計算機を使って課金されまくる[8]のを避けること…であったと。いずれにせよ、かくして ANL にて MPI を用いた並列計算について検討することになったのでした。

最初は、計算機クラスタで、通信頻度がパフォーマンスにどのように影響を与えるかを計測しました。同様の計測を専用の並列計算機で実施したところ、ネットワーク性能が非常に良くて驚いたものです。その後、最終的には VARIANT と呼ばれる輸送ノード法コードを並列化しました。これには専用並列計算機 IBM-SP2 を使いました。最初はどのように並列化すればよいか、右も左も分からなかったのですが、ディスカッションを重ねるうちに方向性が見えてきました。(詳しくは第 4 賞を参照のこと)　この際の経験は、現在の並列計算関連の仕事に大いに役立っています[9]。

---

6 筆者は大阪人なので、「これからは PVM やでぇ！」が実は正しい。
7 通信に関わるアプリケーション・インターフェース(API)のこと。
8 ANL でも課金制度があったのでした。でも、気にせず使えましたが…。
9 交換留学生制度は大変有意義でした。今後も継続されることを強く望みます。

2. 並列計算の実際

さて、筆者の無駄話はこれぐらいにしておいて、いよいよ並列計算の具体的な話について述べていくことにします。ここでは、並列計算とは具体的にどうすればよいか、並列計算を行う際には何に気をつけるべきか、について触れたいと思います。

2.1 問題の分割と通信

最初に、並列計算には何が必要かについて触れましょう。並列計算には、必ずプロセッサ間の「通信」 が必要となります。ここで、簡単な例について考えてみましょう。いま、1 から 100 までの和を計算する場合について考えてみます。これを 1 台の計算機で行う場合には簡単で、図 1 のように処理を行えばよいでしょう。実際には、do ループ一つでできてしまいます。1 台で計算を行う場合、並列計算(parallel computation)と対比して、「**逐次計算**」(serial computation)[10]ということもあります。

次に、これを 2 台の計算機で実施するとどうなるでしょうか？　もう少し格好良くいうと「**2 プロセッサによる並列計算**」ですね。この際、図 1 の処理を 2 台で実行しても何もうれしくありません。計算時間はまったく変化しませんから[11]。計算時間を短縮するためには、1 台当たりの仕事量（計算量）を減らさなくてはなりません。2 台で計算すると、1 台当た

図 1　100 までの和を逐次計算により求める場合

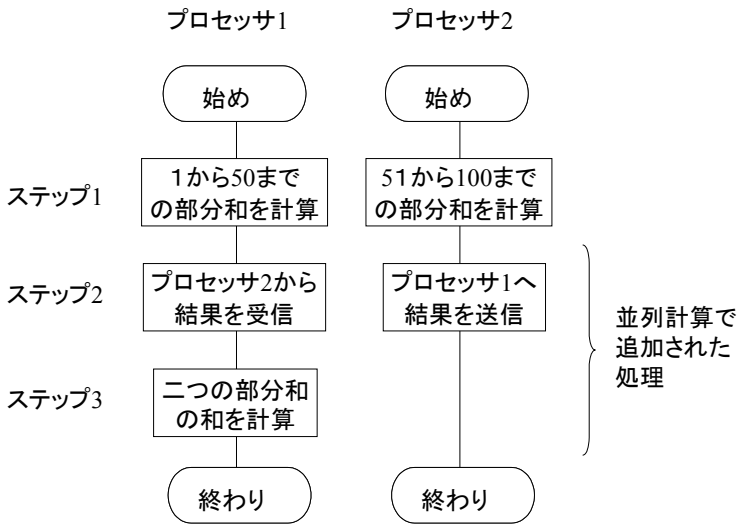


図 2　100 までの和を 2 プロセッサによる並列計算で求める場合

[10] Parallel は複数のものが平行になっている様をあらわし、serial はそれらが団子のように順番にならんでいる様を示しています。そういえば、パソコンにも parallel ポートと、serial ポートがありますね。Parallel ポートではデータ転送に用いる電線が 8 本ありますが、serial ポートでは送受信に各 1 本ずつしか電線はありません。

[11] このことを納得すると、「既存のコードを並列計算機で実行すれば速くなるの？」という質問はナンセンスであることが理解できると思います。

りの仕事は半分で済むはずです。つまり、図 2 のように計算処理を 2 つに分割すればよいのです。

このように、並列計算を行う場合、計算処理をプロセッサ台数分に振り分けなければなりません。これを、解こうとしている「問題の領域」(problem domain)を分解する(decompose)ことから、domain decomposition と呼んでいます。分割の仕方には、いろいろなの方法が考えられます。この例では 1～50 と、51～100 までというように分割していますが、たとえば、奇数と偶数という分割方法も考えられます。このように、個々の分割方法を domain decomposition method (DDM)ということもあります。

少し脱線しますが、炉物理コードでは、どのように domain decomposition することができるでしょうか？例として、Sn 計算コードと、連続エネルギーモンテカルロコードに関して考えてみましょう。Sn コードでは、図 3 に示すように、空間、エネルギー、角度中性子束の角度に関して分割できます。ただし、これらの全てに対して分割するのではなく、空間のみ分割するということが一般的なようです[12]。また、モンテカルロ計算コードに関しては、空間分割を行うものもありますが、バッチあたりの中性子ヒストリー数を分割することが一般的なようです。たとえば、1 バッチあたり 10000 のヒストリー数の計算を 2 プロセッサで行う場合、プロセッサあたり 5000 のヒストリーを追跡するといった具合です。

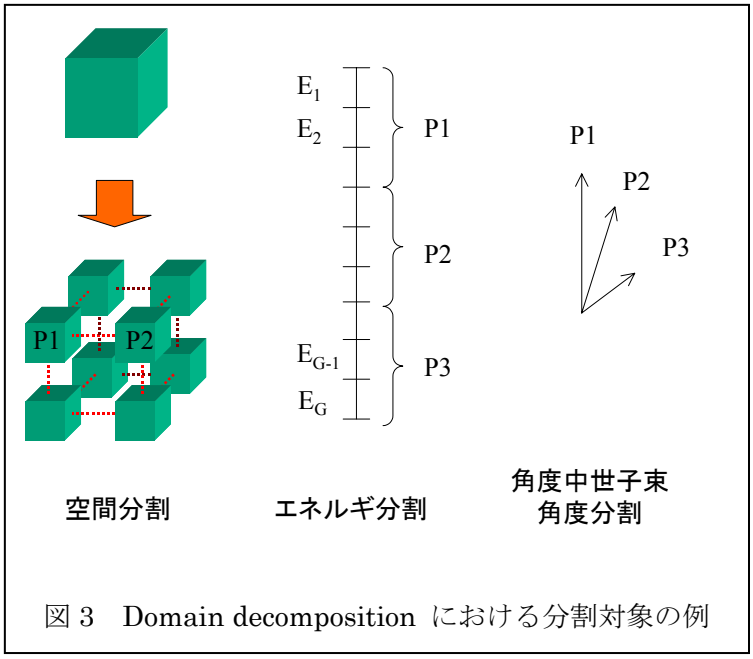


図 3　Domain decomposition における分割対象の例

さて脱線はこのくらいにして、再び図 2 に戻りましょう。並列計算の場合には、逐次計算にはみられなかった、ステップ 2 における「通信」と、ステップ 3 における「集計」が必要となります。本来これらは、**「1～100 の和を計算する」という目的とは何ら関係ない**もので、**ステップ 2 以降の処理に時間がかかってしまっては効率が悪くなってしまう**ということは想像がつくかと思います。この処理時間に関する話は、計算速度について考える

[12] PENTRAN コードは、空間、エネルギー、角度の全てに関して分割可能というツワモノです。

場合に非常に重要となってきますので、次節にて詳しくみてみましょう。

2.2 スピードアップと並列化効率

並列計算を行うと、一般的に、逐次計算よりも速く計算することができます。「一般的に」としたのは、却って遅くなってしまう可能性も多分にあるからです。では、どういうときに速くなって、どういうときに遅くなるのでしょうか？　ここでは、並列化に関する効率の話をいたしましょう。

図 4 には、図 2 における逐次計算と並列計算時の計算時間の内訳を示しています。逐次計算時には、計算機が行うことは計算のみですから、全実行時間は計算時間と同じで T_1 となります。しかし、並列計算時には、逐次計算ではなかった「通信」と「集計」作業が必要となります。計算に必要な時間は半分ですみますから $T_1/2$ となりますが、通信と集計にそれぞれ T_c と T_2 が必要となります。（図 4 では、あわせて「$T_{通信}$」としています。）

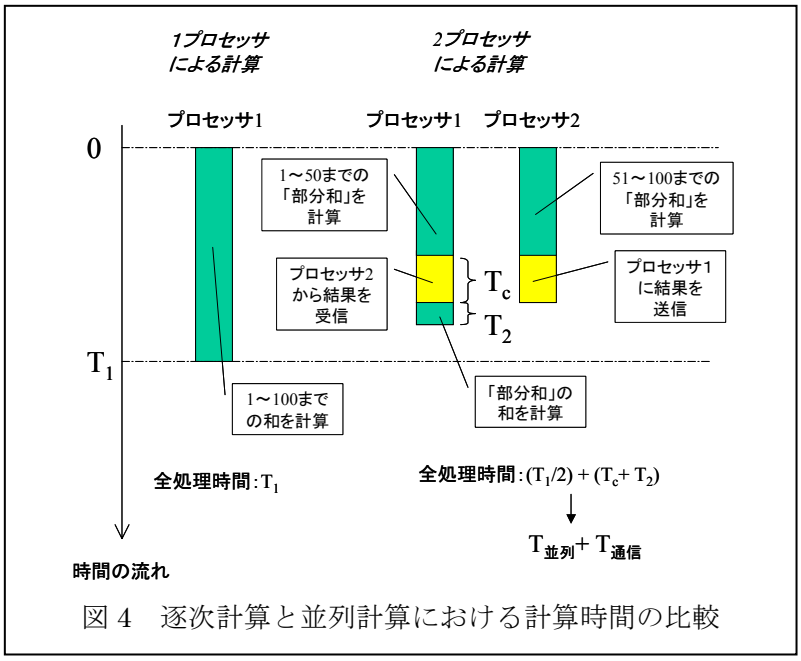


図 4　逐次計算と並列計算における計算時間の比較

ここで、T 通信がまったく無視できるほどに小さければ、全計算時間は逐次計算時の半分で済むわけです。つまり、計算速度が 2 倍となるわけです。逆に、計算とまったく関係のないこの部分が非常に大きくなってしまうと、逐次計算よりも遅くなってしまいます。

どの程度計算が速くなったかをあらわす指標として、(1)式で示される「スピードアップ」があります。

$$Sp = \frac{T_{serial}}{T_{parallel}} \tag{1}$$

これは見てのとおり、逐次計算でかかる時間を並列計算での時間で割ったものです。先ほどの例において、もし「$T_{通信}$」が 0 だとすると Sp は 2.0 となり、理想的なスピードアップとなります。しかしながら、実際には 0 となることはありませんので、2.0 より小さな値と

なります[13]。

スピードアップの例として図 5 を示します。理想的なスピードアップは、プロセッサ数と同じとなり、右上がり 45 度の直線となります。しかし、一般的には通信などのオーバーヘッドがありますので、理想直線よりも下側となります[14]。図中の”Linear”ケースのように、台数に比例してスピードアップが得られる場合は、スケーラビリティー(scalability)があるといいます。しかし、”Not Bad”ケースでは、どこかでスピードアップが飽和してしまうでしょう。また、もっと極端な例としては、”Bad”ケースが挙げられます。プロセッサ数を増やしていくと逆に速度が低下し、ついには逐次計算の時よりも遅くなってしまうでしょう。

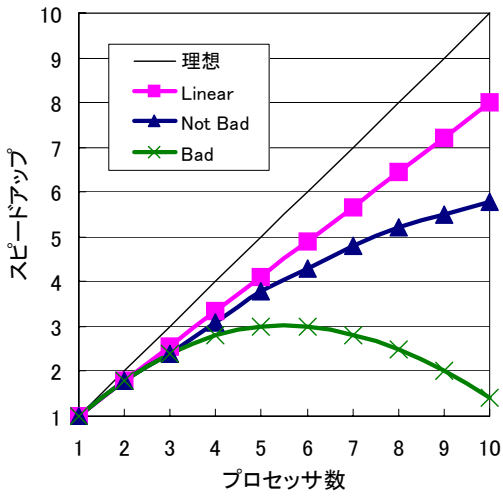


図 5　スピードアップ曲線の例

いま、4 プロセッサで 3 倍のスピードアップが得られるプログラムと、10 プロセッサで 8 倍のスピードアップが得られるプログラムがあったとします。この二つのどちらが優れているでしょうか？　プロセッサ数が異なる場合には、(2)式で定義される並列効率で比較すると分かりやすいでしょう。これは、スピードアップをプロセッサ台数で割ったものになります。

$$Sp = \frac{Sp}{n} = \frac{T_{serial}}{n \cdot T_{parallel}} \tag{2}$$

先ほどの例ですと、3/4=0.75 と 8/10=0.8 ですから、後者の方が効率が高く、優れていることになります。

2.3 並列化率とアムダールの法則

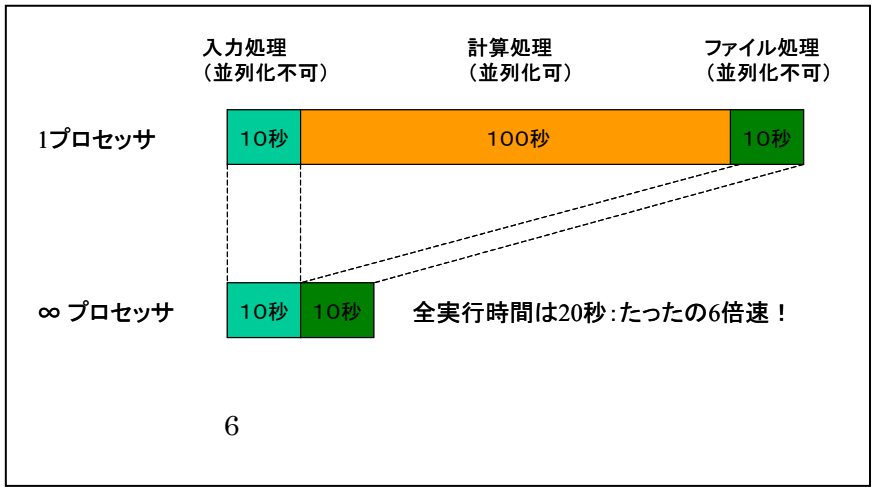
図 5 において、いくつかのスピードアップ曲線がありましたが、これらの違いは何によって起こるのでしょうか？　ひとつの要因は、前節で説明したとおり、通信に関わる時間「$T_{通信}$」が有限であるということです。つまり、本来計算とは関係のない処理があるのですから、どうしても完全に理想的なスピードアップとはならないでしょう。もうひとつの

---

[13] スカラー計算機の場合、実際には問題を分割することによりキャッシュ適合性があがり、2.0 を超えることがあります。（これを「キャッシュ効果」と呼びます。）

[14] キャッシュ効果により、理想直線を越える”super-linear”となることもあります。

重要な要因が、「プログラムの全実行時間のうちどの程度が並列化されているか」を表す「並列化率」あるいは”parallel fraction”と呼ばれる因子です。



たとえば、入力処理、計算、ファイル書き出しの 3 つのパートからなっている並列プログラムがあったとします。図 6 に示すように、1 プロセッサで実行した場合、それぞれの実行時間が、10 秒、100 秒、10 秒であったとします。仮に、計算部分が理想的に並列化できて、無数にあるプロセッサで一瞬のうちに計算ができたとしましょう。ただし、入力処理とファイル書き出しは並列化が無理だとします。すると、全体の計算時間は、限りなく 10 秒+10 秒=20 秒に近づくことになります。あれ？ちょっと待ってください。プロセッサは 1000 とか、10000 とか多く使っても、たった 6 倍程度のスピードアップしか得られないではありませんか！ これが、有名な「アムダールの法則(Amdahl's Law)」が示唆することです。 プログラムの全実行時間 T のうち、並列化可能な部分の割合を α とすると、n プロセッサ時の最大スピードアップは(3)式で表されます。

$$Sp_{\max} = \frac{T}{\frac{\alpha T}{n} + (1-\alpha)T} = \frac{1}{\frac{\alpha}{n} + (1-\alpha)} \tag{3}$$

では、α や n をいろいろ変えて、最大スピードアップがどのように変化するかを見てみましょう。（図 7） α が 0.9 程度では、すぐにスピードアップが頭打ちになることが分かります。実用的なプログラムとするためには、α が 0.99 以上は欲しいところです。特に大規模計算の場合には、できるだけ α を 1 に近づけることが重要となります。このあたりが並列計算の難しいところであり、同時に挑戦し甲斐のあるところです。

さて、この並列化率について少し考察してみます。先日稼働した日本が誇るスーパーコンピュータ「地球シミュレータ」は、Linpack と呼ばれるベンチマーク計算において、ピーク性能 40Tflops に対して、実効性能 35.61Tflops を記録しました。これまで世界最高速

だった米国のマシンから一気に 5 倍以上差をつけて、現在ダントツ世界一の性能です[15]。このときのプロセッサ数は5104だったので、(3)式からαを逆算してみると、実に 0.999976 となり、非常に高い並列化率を達成していることがわかります。逆に言うと、この程度の並列化率がないと、大規模並列計算環境で高いスピードアップ（並列効率）を得ることが難しい訳です。実際の気候シミュレーションでは、26Tflops 程度の実効速度があったそうですが、それでもαは 0.9999 (four nines)程度となります。これは、プロセッサ数が非常に多いときの話であり、10 台程度ではそこまで気にすることは無いでしょう。

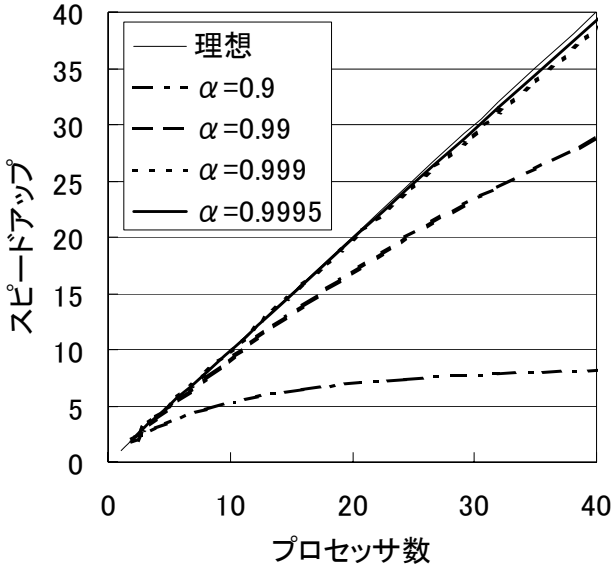


図 7　並列化率とスピードアップの関係

3. 並列計算をやってみよう！

さて、これまでに、「並列計算では問題を分割してプロセッサに仕事を分配する」ということをお話ししました。また、高いスピードアップを得るためには、「通信に伴うオーバーヘッドを小さくする」ことや、「並列化する部分を多くする」必要があることも述べました。では具体的にはどうすれば良いのでしょうか？　本節では、少し具体的に見ていきましょう。

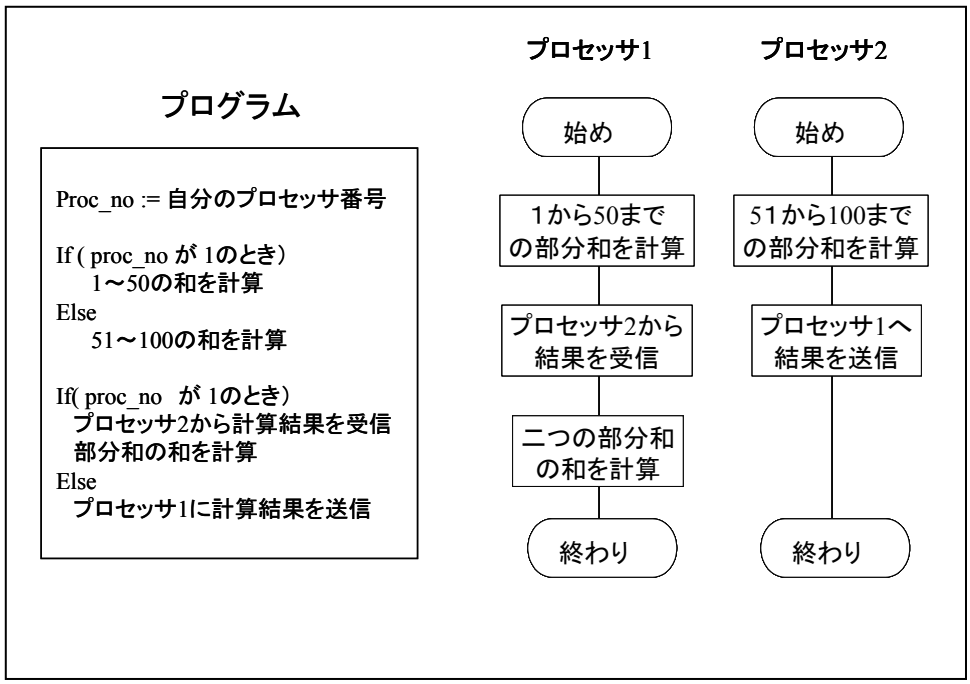
3.1 SPMD プログラミングモデル

並列プログラムを作るといっても、まだイメージが湧きにくいと思います。これは、筆者も同様でした。並列プログラムというと、各プロセッサで走らせるプログラムを別々につからないといけないのでしょうか？　いいえ、通常はそういった複雑なことをする必要はありません。一つのプログラムだけで良いのです。

並列計算を実行する際には、各プロセッサには固有の番号が割り当てられています。ですから、プログラムの中には、自分がプロセッサ 1 だった場合はどうする、プロセッサ 2

[15] 米国はかなりショックだったらしく、ソ連に人工衛星開発レースで負けたときの”Sputnik shock”にもじって、”computonik shock”と言っています。

だった場合はどうするというように場合を分けて処理を記述しておけばいいわけです。そうすると、作成するプログラムは一つだけで良いことになります。ただし、各プロセッサ上では、異なったデータが取り扱われることとなります。（図 8）



このようなプログラミングモデルを、SPMD(Single Program Multiple Data)と呼びます。並列処理を行う場合、99.9%までは SPMD モデルでことが足りるといっても過言ではないと思います。

3.2 並列計算機と並列ライブラリ

並列計算を行うには、並列計算機が必要です。では、並列計算機ってどんなのでしょうか？　詳しくは 2 章で話がありますが、大きく分けて二つの種類があります。一つは共有メモリ型の計算機、もう一つは分散メモリ型の計算機です。身近な例で言うと、前者は一つの筐体に複数の CPU が乗っている、特にサーバーマシン等が良い例です。後者は、ネットワークで接続されたパソコンや EWS の集合体と考えれば良いでしょう。これらの違いは、並列ライブラリを使うことによって、特に意識する必要はなくなります。

並列ライブラリとは、並列プログラムを書く際に、プロセッサ間の通信等の処理を一手に面倒を見てくれる便利な関数群です。先ほど、並列計算を実行する際には各プロセッサは固有の番号を持つと述べましたが、実はこのようなことも並列ライブラリが前もって準備しているからなのです。並列ライブラリには、先述の PVM や MPI といったものが挙げられます。特に、MPI は業界標準となりつつあるので、こちらを勉強されることをおすすめします。MPI を用いた並列プログラムの作成については 3 章で解説がありますのでご覧ください。

4. さいごに

本章を読んで、並列計算ってやってみたいけど、難しそう」と思っていたあなたも、並列計算とはどういったものかというすこし具体的なイメージが湧いたでしょうか？　確かに、並列計算を行うためには、それなりの手順をプログラムしてやる必要があります。しかし、並列ライブラリを使うとそれほど難しいことではありません。身近にネットワークで接続された計算機が何台かあれば、今すぐにでも並列計算を行うことができます。「習うより慣れろ」の感覚で、一度トライされてみてはいかがでしょうか？　並列計算の世界への皆さんのご参加をお待ちしております。

並列計算は、計算を速くするための便利な道具です[16]。最後に、筆者からのメッセージ[17]を添えて第１章の幕を下ろしたいと思います。

Parallel, it works!

(並列計算はうまくいく！)

[16] 気をつけないと、手段ではなくて目的になってしまう場合があります…。
[17] 実は、筆者の E-mail 用のシグネチャーにこの文句が添えられています。