並列計算理論

Author: 小倉 康睦

Date: 17/9/2017

並列計算理論

「並列計算とは何か？」この問いに対する答えを持つ人間はおそらくこの文章を読む必要がない。あなたはどうだろうか。私たちにとって、眼前の個々の問題を並列化して解く方法を見出すことは比較的容易い。なぜならば私たち人間は社会性動物としての本能を有しており、個人では到底解決できない大きな問題を協力と役割分担によって解決することで文明を発展させてきたからだ。

「眼前の個々の問題を並列化して解く方法を見出す」という操作は「並列計算とは何か？」という問いに対する帰納的なアプローチと言える。「あの問題は並列化できる」「この問題は並列化できない」といったように、問題を提示されなければ議論は展開されない。これは私たちの知的レベルの現在における限界を示している。
「個々の計算機では到底解決できない大きな問題を協力と役割分担によって解決する」ことは、並列計算の本質と言えるだろうか？だとしたら分散型の並列計算におけるフォールトトレランスはどう説明するのだろう。フォールトトレランスに重点を置いた並列計算は、個々の計算機でも十分に解決可能な問題をあえて多くの計算機に分配することによって実現されることもある。結局、現在実現している並列計算技術から私たちが垣間見ることができるのは並列計算機のある側面に過ぎないのではないだろうか？私たちはそれを議論するに足るだけの知識を持ち合わせてはいないのだ。

私たちは次なるステージとして、並列計算を演繹的に理解したい。すなわち「並列計算とは何か？」という問いから「並列計算によって何ができるのか」を導きたい。このアプローチはまだ完成されていない途上のものであることは否定しない。今一度問うことにしよう。「並列計算とは何か？」この問いに対する答えを持つ人間はおそらくこの文章を読む必要がない。あなたはどうだろうか。

並列計算とは

計算とは

並列計算と一口に言ってもその実現方法は多岐に渡る。私たちは演繹的な議論を可能とするために並列計算とはなんたるかを定義し直さなければならないが、そのためにはまず、計算とはなんたるかを定義せねばなるまい。

計算(calculation)あるいはプログラムとは何らかの目的を持って実行される、**情報(data, information)**に対する操作あるいは加工である。一般に計算Fはデータの集合Dを考えたとき、入力$d \in D$の出力$d' \in D$への写像$F(d) \to d'$である。ここで=ではなく→を用いたのは、計算が必ずしも可逆とは限らないからである。

計算Fは一般により小さな計算fの**集積(accumulation)**に分解することができる。計算Fが有限なn個の計算$f_1, f_2, \dots, f_n$によって構成されている場合、これを

$$F = Accum\left\{f_1, f_2, \dots, f_n\right\} \tag{1}$$

と書き、右辺をFの**集積分解(accumulative factorization)**という。このとき$f_i : i \in \{1,2,\ldots,n\}$を$F$の**集積因子(accumulative factor)**といい、nを**分解量(amount of factor)**という。一般に計算Fの集積分解の仕方は1通りではなく複数存在する。たとえば任意の計算FはF自身の集積であるから、計算の集合をΦで表すと次の式が恒等的に成立する。

$$\forall F \in \Phi, F = Accum\{F\} \tag{2}$$

計算fがそれ以上小さな計算の集積に分解できないとき、そのfを**アトミック(atomic)**な計算であるという。アトミックな計算の集合をΦ_{atom}で表す。アトミックな計算fはそれ自身の集積以外ではありえず、集積分解が次のただ一通りに定まる。

$$\forall f \in \Phi_{atom}, f = Accum\left\{f\right\} \tag{3}$$

(1)の式は、単にFの構成要素を列挙したにすぎないため、集積の操作$Accum$は与えられた複数の操作に関して具体的な実行順序までは要請しない。したがって$Accum$内に列挙された操作を適当な順番で実行した場合、その計算がFに一致する保証はない。厳密に実行順序まで指定して計算を分解する場合は**因果的連鎖(sequence)**の操作を用いる。因果的連鎖は単に連鎖(chain)ともいう。

計算fが実行され始める時間を$t_{start}(f)$、実行が終了する時間を$t_{end}(f)$で表す。計算Fが$f_1, f_2, \ldots, f_n$のn個の計算の因果的連鎖に分解できるとき、

$$F = Seq\left\{f_1, f_2, \ldots, f_n\right\} \tag{4}$$

と表し、右辺をFの**連鎖分解(sequential factorization)**という。このとき$f_i : i \in \{1,2,\ldots,n\}$を$F$の**連鎖因子(sequential factor)**という。任意の$f_i : i \in \{1,2,\ldots,n-1\}$について

$$t_{end}(f_i) < t_{start}(f_{i+1}) \tag{5}$$

が成り立っている。また、このことから明らかに

$$\begin{cases} t_{start}(F) = t_{start}(f_1) \\ t_{end}(F) = t_{end}(f_n) \end{cases} \tag{6}$$

が成立する。一般にFの連鎖分解の仕方は1通りではなく複数存在する。たとえば任意の計算FはF自身の連鎖であるから、計算の集合をΦで表すと次の式が恒等的に成立する。

$$\forall F \in \Phi, F = Seq\{F\} \tag{7}$$

これは(6)式を満たす。また、アトミックな計算fはそれ自身の連鎖以外ではありえず、連鎖分解がただ一通りに定まる。

$$\forall f \in \Phi_{atom}, f = Seq\left\{f\right\} \tag{8}$$

連鎖分解の仕方は複数通り考えられるが、その中でも同一とみなせる分解が存在する。Fが$f_i : i \in \{1,2,\ldots,n\}$に連鎖分解できるとき、適当な$i : i \in \{1,2,\ldots,n-1\}$を選んで、次の式が成立するとみなす。

$$Seq\left\{f_1, f_2, \ldots, f_n\right\} = Seq\left\{Seq\left\{f_1, \ldots, f_i\right\}, Seq\left\{f_{i+1}, \ldots, f_n\right\}\right\} \tag{9}$$

これは新たに$F_1 = Seq\left\{f_1, \ldots, f_i\right\}$、$F_2 = Seq\left\{f_{i+1}, \ldots, f_n\right\}$と置き直せば(6)式を満足する。

また$i : i \in \{1,2,\ldots,n-1\}$を選んで次の式

$$Seq\left\{f_1, \ldots, f_i, f_{i+1} \ldots, f_n\right\} = Seq\left\{f_1, \ldots, f_{i+1}, f_i \ldots, f_n\right\} \tag{10}$$

が成立するとき、f_iとf_{i+1}は**同順位の因果**であるといい、どちらを先に実行しても結果が変わらないことを示している。

集積分解と連鎖分解を特に区別しない場合は単に**分解(factorization)**といい、このとき集積因子と連鎖因子も区別せずに単に**因子(factor)**という。

ここまでで扱ってきた分解量nは有限であった。nが無限である場合はどうなるだろうか。一般に計算Fをどのように分解しても可算無限個の因子に分解される場合、その計算Fは停止しない。このことを示すために**計算機(computer)**を定義する。

計算機Cは**状態(statement)**Sを持つ。Cが計算Fを実行して状態S'に遷移し、残りの計算がF'となるとき次のように表す。

$$F : S > C \to F' : S' \tag{11}$$

計算機Cは有限個の因子で構成される計算Fを有限な時間内に実行することができるものとする。実行にかかる時間を$T(C, F)$とおくと次の式が成り立つ。

$$T(C, F) = t_{end}(C, F) - t_{start}(C, F) \tag{12}$$

実行する計算機Cが明らかな場合、Cを省略して次のように書いてもよい。

$$T(F) = t_{end}(F) - t_{start}(F) \tag{13}$$

また、明らかに因子の多い計算では、因子の多い計算のほうがより計算時間がかかるので、

$$\forall f_1, \forall f_2, T(Accum\left\{f_1, f_2\right\}) \geq T(Accum\left\{f_1\right\}) \tag{14}$$

が成り立つ。等号が成立する条件は、この計算機がf_1を処理するときの余剰の計算能力でf_2を十分に処理できるときである。すべてのアトミックな計算に対して次の不等式が成り立つとする。

$$\forall f \in \Phi_{atom}, T(f) > 0 \tag{15}$$

「計算を構成する因子のうちでもっとも小さい計算であっても、それを処理するのにはわずかでも時間がかかる」というのが(15)式の主張するところである。

可算無限個の因子で構成される計算を考えるとき、その因子の中でもっとも計算時間の短い計算をf_{min}とおく。(15)式より

$$T(f_{min}) = \epsilon > 0 \tag{16}$$

を満たす実数ϵが存在する。したがって、この計算機が一度に処理できる、有限個の因子で構成される最大の計算Fについて

$$T(F) \geq \epsilon > 0 \tag{17}$$

が成り立っている。この計算機がM回の計算ですべての因子を網羅して計算を完了すると仮定すると、そのときかかる時間はϵMで表すことができる。分解量nが可算無限となるとき、$M > 0$を満たす任意のM回目の計算を完了したとき、$N > M$なるNが存在してN回目の計算を行っても因子は網羅されない。したがって$\tau > 0$を満たす任意の時間的刻限τを選んだとき、$\epsilon N > \tau$となるNが存在するので、分解量nが可算無限となる計算は停止しない。計算の分解量nは、計算を実行する前から確認できるとは限らない(停止性問題)。

並列計算とは

私たちはこれから並列計算とは何かを定義していくことになるが、現在実現している並列計算技術がその定義から外れてはならない。また未来的に開発される並列計算技術も表現しうる汎用性を備えていなければならない。したがって採用する定義は抽象的なものとなる。

並列計算を定義する前に、ここまでに述べてきた計算の定義を用いて、どのように並列計算が実現されうるのかを概説しておく。(10)式で説明される「同順位の因果」が並列計算の鍵となる。同順位の因果とはすなわち「実行順序に依存しない」計算を表している。したがって(10)式の左辺は次のように書き直すことができる。

$$Seq\left\{f_1,\dots,f_i,f_{i+1}\ \dots,f_n\right\} = Seq\left\{f_1,\dots,Accum\left\{f_i,f_{i+1}\right\}\dots,f_n\right\} \tag{18}$$

実行順序に依存しないということは、このf_iとf_{i+1}を同時に実行できる可能性を秘めている、ということを示している。ただし必ずしも同時に実行できるとは限らないことには注意が必要である。仮にf_i,f_{i+1}が指定したメモリに1を足しあげる操作だったとしよう。f_i,f_{i+1}が同じメモリを指定していた場合、この2つの計算を同時に実行することは悲劇を招く。多くの実装ではこれは未定義動作であり、メモリの内容を破壊してしまうだろう。これは実行順序にはよらないが、同時に実行することのできない例となっている。

計算Fが有限なn個の計算$f_1,f_2,\dots,f_n$によって構成されている、すなわち

$$F = Accum\left\{f_1,f_2,\dots,f_n\right\}$$

を満たし、なおかつ$f_1,f_2,\dots,f_n$が同時に実行できるとき、これを

$$F = Parallel\left\{f_1,f_2,\dots,f_n\right\} \tag{19}$$

と書き、右辺をFの**並列分解(parallel factorization)**という。このとき$f_i : i \in \{1,2,\dots,n\}$を$F$の**並列因子(parallel factor)**という。

計算Fが並列分解可能で、その並列因子を同時に実行できる計算機を**並列計算機(parallel computer)**という。並列計算機Pは接続された複数の計算機であり、PがN個の計算機$C_1,C_2,\dots,C_N$によって構成されているとき、これを

$$P = Connect\{C_1,C_2,\dots,C_N\} \tag{20}$$

で表す。$C_1,C_2,\dots,C_N$を並列計算機Pの**メンバ(member)**といい、メンバの数Nを**メンバ数(number of member)**という。各メンバは並列計算機であってもよい。メンバである計算機がいずれも並列計算機ではないときのメンバ数を**計算機量(amount of computer)**といい、並列計算機が同時に実行できる並列因子の数の上限を表す。並列計算機でない計算機とは、計算機量が1である計算機のことであり、これを**逐次計算機(sequential computer)**という。計算機量がNである並列計算機Pで、計算Fの並列分解の分解量nが$n > N$となるとき、この並列計算機では計算Fを一度の計算で完了することはできない。

並列計算機は有限個の並列因子で構成される計算$F = Parallel\left\{f_1,f_2,\dots,f_n\right\}$を有限な時間内に実行することができるものとする。実行にかかる時間を$T(F)$とおくと次の式が成り立つ。

$$T(F) \geq \max\left\{T(f_i)\right\} \tag{20}$$

Fを並列分解すれば無条件に並列計算となるわけではないことには注意されたい。*Parallel*はあくまでも同時実行可能性を付加した*Accum*に過ぎないことを思い出せば、逐次計算機で実行した場合はただの順不同の逐次実行となるだけである。また、並列分解不能なFを並列計算機で実行しても同時実行可能なセクションが存在しないため、並列計算とはならない。並列計算は、並列分解されたFを並列計算機によって実行することで初めて成立する。

通信(communication)は並列計算において重要なものとなる。通信とはある計算機から別の計算機へ何らかの情報を伝達することである。並列計算においては次の重要な性質が成り立つ。

- 並列計算に加担するすべてのメンバは、少なくともひとつの他のメンバと必ず何らかの通信を、少なくとも1回行なう。 ... (21)

この性質が成り立つとする根拠を以下に示す。並列計算に加担するメンバは次の2つのフェーズの少なくとも一方を必ず実行し、その際に何らかの通信が発生する。

1. プログラム分配フェーズ
2. 計算結果集積フェーズ

2つ以上の計算機が並列計算に加担しうる特定のプログラムを偶然所持していることは(なきにしもあらずだが)一般的に確率0の起こり得ない事象とみなしてよい。並列計算機$P = Connect\ \{A, B\}$において並列計算を行いたい場合には次の3通りの方法のいずれかによってプログラムを共有しなければならない。

1. Aが自身の所持するプログラムをBへ送信する
2. Bが自身の所持するプログラムをAへ送信する
3. 第三者である計算機Cが何らかの方法でA, Bへプログラムを送信する

これがプログラム分配フェーズである。 3番目の方法ではA, Bは直接的には通信していないことになるが、Cは明らかに並列計算に加担したことになるので、$P = Connect\ \{A, B, C\}$と定義し直すことで(21)の性質を満たし、この性質はメンバ数が一般のNの場合にも成り立つ。

2つ以上の計算機が並列計算に加担して計算を分担し、ある一方がもう一方の計算結果を正確に知りたい場合は通信によって問い合わせるしか方法はない。並列計算機$P = Connect\ \{A, B\}$において並列計算の結果を集積したい場合には次の3通りの方法のいずれかによらねばならない。

1. Aが自身の所持する計算結果をBへ送信する
2. Bが自身の所持する計算結果をAへ送信する
3. 第三者である計算機Cが何らかの方法でA, Bの計算結果を中継する

これが計算結果集積フェーズである。 3番目の方法ではA, Bは直接的には通信していないことになるが、Cは明らかに並列計算に加担したことになるので、$P = Connect\ \{A, B, C\}$と定義し直すことで(21)の性質を満たし、この性質はメンバ数が一般のNの場合にも成り立つ。

すべてのメンバはプログラム分配フェーズと計算結果集積フェーズの少なくとも一方を持つが、必ずしも両方を持つとは限らない。たとえば相手がどのような計算をしたかは未知であるが、その結果のみを知りたい場合というのは統計分析やシステムの解析ではよくあることである。着目している並列計算プログラムとは源流の異なるブラックボックスを観測するときはプログラム分配フェーズを仮定するのは妥当ではなく、計算結果集積フェーズのみを持つとする。また、たとえば相手に依頼した計算の結果に興味がない場合、計算結果集積フェーズを仮定する必要はない。

ここまでの定義を用いて、一例として「同期システム機能付道路工事用点滅灯」を並列計算機と見なして分析する。道路工事用の点滅灯は車両が工事現場に誤って侵入しないために重要な役割を果たしている。個々の点滅灯は接続されていないことが多いため、スイッチを入れたタイミングやクロックの品質のバラつきのために同じタイミングで明滅することはない。これを改良したのが同期システム機能付道路工事用点滅灯である。この点滅灯は電波時計の信号を受信して明滅のタイミングを決めているため、互いに接続していなくても点滅の同期を取ることができる。これら個々の点滅灯は接続していないが、工場出荷時に同じプログラムを書き込まれている。また、点滅のタイミングは電波時計との通信によって制御されている。したがって、これら点滅灯、工場、電波時計塔を含めてひとつの並列計算機$P = \left\{\textbf{点滅灯}, \textbf{工場}, \textbf{電波時計塔}\right\}$とみなした場合、すべての点滅灯は、工場と電波時計塔によってプログラム共有フェーズで結合している。個々の点滅灯が点滅した結果については工場や電

波時計は興味を持たないため、計算結果集積フェーズは持たない。計算結果集積フェーズで点滅灯と結合されているのは人間である。人間は点滅灯の点滅という計算結果を受け取って、点滅灯への必要以上の接近を回避する。このとき並列計算機の定義域は$P = \{\textbf{点滅灯}, \textbf{人間}, \textbf{工場}, \textbf{電波時計塔}\}$に拡張される。

たかが信号灯を点滅させるくらいのことが並列計算と呼べるのか、こじつけではないかという指摘はしごくまともなものである。しかし私たちが知っているのはあくまでも「並列計算に加担するすべてのメンバは、少なくともひとつの他のメンバと必ず何らかの通信を、少なくとも1回行なう」という事実のみであって「どのような通信を行ったら並列計算になるのか」まで言及できるほど理論は完成されていない。それまで私たちはなるべく抽象的で広範な現象を記述できる定義に従うしかない。最近、アフリカの社会性肉食獣リカオンがクシャミによって群れが狩りに出るかどうかの民主的投票を行なっているのではとの報告がある。彼らのこの行動を、群れが狩りの効率を最適化するための並列計算だとみなすのならば、その通信に使われたのは「たかがクシャミ」である。私たちはおそらくたかがクシャミでさえ通信の定義から外してはならないのだ。

ここまでのことを一度整理して次の議論に移るとしよう。並列計算機で並列計算を行うことを、次のように定義する。

1. 計算機が2以上のN個のメンバから成り、$P = Connect\{C_1, C_2, \ldots, C_N\}$と表せるとき、この$P$を並列計算機と呼ぶ。
2. 並列計算機が並列分解された計算Fを実行するとき、並列計算を行うという。
3. 並列計算に加担するすべてのメンバは、少なくともひとつの他のメンバと必ず何らかの通信を、少なくとも1回行なう。

なお、このときの通信とは「ある計算機から別の計算機へ何らかの情報を伝達すること」であり、着目するシステムに応じて定義する。