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(calculator)**を定義する。

計算機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$T(F)$とおくと次の式が成り立つ。

$$T(F) = t_{end}(F) - t_{start}(F) \tag{11}$$

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、

$$\forall f_1, \forall f_2, T(Accum\left\{f_1, f_2\right\}) \geq T(Accum\left\{f_1\right\}) \tag{12}$$

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。

$$\forall f \in \Phi_{atom}, T(f) > 0 \tag{13}$$

「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(13)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。(13)式より

$$T(f_{min}) = \epsilon > 0 \tag{14}$$

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

$$T(F) \geq \epsilon > 0 \tag{15}$$

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。

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

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。