［研 究 論 文］

e-Gov法令検索サイトにおける 個別条文アクセス処理の並列化について

An Experimental Parallel Processing of the Database Access of Articles from e-Gov Laws and Regulations Search Website

関本大樹
Taiju Sekimoto

久留米大学 法学部
Faculty of Law Kurume Univ.

要旨：

e-Gov 法令検索サイトにおける個々の条文データのアクセスについては、一定のターンアラウンド・タイムを要するが、特に規模の大きな法令については、アクセスに成功した場合と成功しなかった場合のターンアラウンド・タイムの差が大きい傾向がある。そこで、枝番号付の条文を悉皆的にツリー探索した場合について、並列処理によってどの程度、当該アクセスのスループットを向上できるのか実際に確認してみることとした。

その結果、スループットを 5 倍ないし 20 倍程度に向上できることが分かった。しかし、当該実験結果を踏まえても、個別の条文について、その枝番の有無等が分からない現在の API については、非効率的であるとも考えられることから、その仕様を改善することを提案したい。

Abstract:

It requires some turnaround time to access each article from e-Gov Laws and Regulations Search Website. And especially in the case of large laws or regulations, the difference of throughput between the successful and unsuccessful access tends to be larger. Therefore, I decided to experiment in the effect of parallel processing in order to make the throughput more efficient in the tree search application of sub-numbered articles.

As a result of the above experiment, total throughput by multiprocessing can be 5 to 20 times larger than by single processing. In spite of such possible improvement, from the point of view of efficiency, the current API without properties of sub-numbered articles should be highly recommended to be improved sooner or later.

【目次】

1. はじめに
2. 実験環境
   2.1 開発環境等
   2.2 使用パソコン
3. 条文調査フォームの概要
   3.1 本フォームの各項目
   3.2 各ボタンの動作プログラム及び関連事項
4. 実験結果
   4.1 調査対象法令
   4.2 ターンアラウンド・タイム
   4.3 逐次処理結果
   4.4 並列処理結果
5. e-Gov 法令検索サイト用 API の改善提案
6. おわりに
（別添資料）条文調査フォームのソースコード

1. はじめに

e-Gov 法令検索サイト（https://elaws.e-gov.go.jp/。以下、単に「検索サイト」という。）では、指定した現行法令について一般のユーザー・プログラムからその全文を取得するためのAPI（「法令取得 API」と呼ばれる。）が提供されているが、租税法分野でいえば主要法令である所得税法、租税特別措置法、地方税法などのようなデータサイズの大きい一部の法令（現時点で 31 法令）については、タイムアウトを防ぐため、当初、当該 API の対象外とされており、利用者による一括ダウンロードによって対処することが予定されていた(1)。

他方、指定した法令の該当条文を個々に取得するための API（「条文内容取得 API」と呼ばれる。）が提供されており、これについては、上記のような制限がないものの、個々の条文の直前の条文や直後の条文を参照する機能が提供されていない。そのため、条文には、枝番付のものがある（なお、枝番付でない条文を以下「基本条文」という。）ことから、当該 API によって、ユーザー・プログラムからリアルタイムで全ての条文を参照することは容易ではなかった。ただし、各条文の枝番については、次のような規則性のあるツリー構造が認められるため、やや迂遠ではあるものの、当該構造に着目すれば、上記の条文内容取得 API を用いて、ほとんどの法令について、附則以外のすべての条文を取得することが可能である。

例えば、ある法令について「第３条の２」という条文の次の条文が仮にあるとすれば、当該条文は、必ず①「第３条の２の２」、②「第３条の３」又は③「第４条」の３通りのうち、いずれか一つに限られることになる。なお、ある条文が改正により削除されている場合であっても、当該条文が削除されていること自体が当該条文に記載され、原則として、上記のような連続性が失われることはない。

したがって、上記の可能性のある３通りを全て順番に試行的にアクセスしてみれば、次の条文に辿り着くことができるわけである。しかし、当然、その試行の際には、ほとんどのアクセスについて不成功（HTTP レスポンスコード「NotFound」）となるが、その際のターンアラウンド・タイムが、当該法令により異なるものの、成功した場合（同「OK」）に比べ、大変長くなる場合が多い。例えば、所得税法の場合、アクセスが成功した場合には、ターンアラウンド・タイムは 0.7～0.8 秒ほどであるが、不成功の場合には、４～５秒となる傾向がある。そのため、同法の条文数については基本条文のみでも 243 条あることから、全条文（299 条）を確認するために、28 分間ほどを要することになる。

このようにサーバー・アクセスのターンアラウンド・タイムが処理上のネックとなる場合には、それに対する直接的な対処方法の一つとして、並列処理を行うことが考えられる。そこで、筆者は、米国 Microsoft 社の開発環境である Visual Studio 2019 を用いて個別条文のアクセス処理の並列化を実験的に行ってみることとした。本稿では、その具体的な方法や実験結果について概説するとともに、上記検索サイトに係る API の今後の機能改善策について提案することとしたい。

(1) ただし、現在の検索サイトでは、リニューアルが行われたことにより、例外とされていたこれらの法令についても法令取得 API のサービス対象とされている。なお、その詳細については、本稿末尾の【追記】を参照されたい。

2. 実験環境

2.1. 開発環境等

プログラムの並列処理やサーバー・アクセスなどの非同期処理については、そもそもMicrosoft Windows などのオペレーティング・システムの基盤となる処理技法であるものの、これまでは、アプリケーション・プログラム開発に容易に応用するための基盤が整備されていなかった。しかし、昨今では、今回の実験で用いたVisual Studio 2019（https://visualstudio.microsoft.com/ja/）などの開発環境を利用することにより、そのような処理技法を大変容易に実現することが可能になってきている(2)。このような流れの背景としては、マイクロプロセッサの多コア・多スレッド化が飛躍的に進行してきていることが挙げられよう。

筆者は、これまで主にMicrosoft 社が開発したVisual Basic（ないしVisual Basic for Applications）を開発用言語として利用してきているが、当該言語は、Visual Studio 2019 でも利用可能なため、今回のシステム開発においても利用することとした。

なお、上記開発環境には並列処理のアプリなどの動作も柔軟に確認することができる強力なデバッガが用意されており、しかも、一定の条件の下で無償により利用可能である。大変良い時代になったものである。

なお、実験を行うためのフォーム・アプリである「条文調査フォーム」（下記 3 参照）は、従来型のWindows フォーム・アプリ(3)ではなく、フォーム自体の機能とフォームによって実行される動作とを完全に分離することを可能としたとされる、より新しい Windows Presentation Foundation（WPF）フォーム・アプリとして作成してみることとした。

2.2. 使用パソコン

念のためパソコンの処理能力の違いによる実験結果への影響の度合いについても検討するため、実験には、次の２種類のパソコンを使用することとした。

- Intel Core i7-8700 CPU @3.20GHz（６コア・12 スレッド）のデスクトップ・パソコン（以下、単に「デスクトップ」という。）
- Intel Core i5-7Y54 ＠1.20GHz（２コア・４スレッド）のノート・パソコン（以下、単に「ノート」という。）

(2) 並列処理を容易にする現在の仕組みは、「タスク並列ライブラリ（TPL）」と呼ばれるが、Visual Basic 2010 で採用されており、また、非同期プログラミングに用いられる仕組みである「Async/Await 構文」については、Visual Basic 2012 で採用されている。尾崎義尚「Visual Basic 2010 の新機能」『特集：VB10 概説』（令和２年３月 31 日現在）https://www.atmarkit.co.jp/fdotnet/chushin/vb2010features_01/vb2010features_01_04.html、鈴木孝明「第１回　.NET 開発における非同期処理の基礎と歴史」『連載：C#5.0&VB 11.0 新機能「async／await 非同期メソッド」入門』（令和２年３月 31 日現在）
https://www.atmarkit.co.jp/fdotnet/chushin/masterasync_01/masterasync_01_01.html など参照。

(3) Windows フォームは、「後発のデスクトップアプリケーションフレームワークである WPF に比べると、マルチタッチや DPI Aware などに標準で対応していないなど、最新の技術動向は反映されにくい傾向にある」とされている。（令和２年３月 31 日現在）https://ja.wikipedia.org/wiki/Windows_Forms 参照。

3. 条文調査フォームの概要

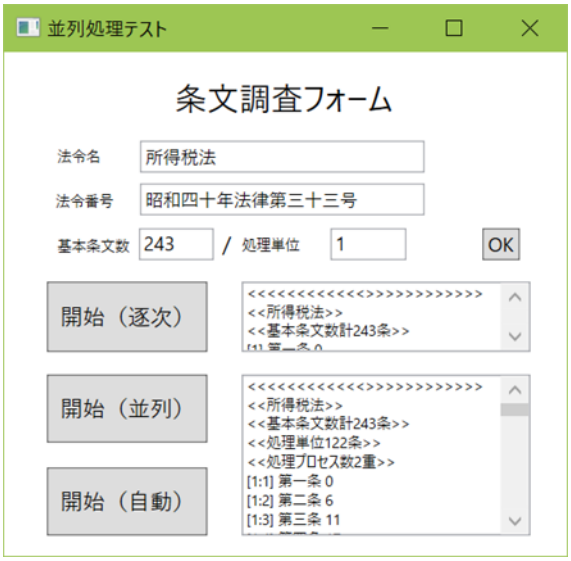

図 1　条文調査フォーム

3.1. フォームの各項目

(ア)「法令名」テキストボックス

条文数を調査する対象となる法令の名称（例えば、「所得税法」、「民法」など）を指定するためのテキストボックスである。

(イ)「法令番号」テキストボックス

アクセス対象となる法令については、API の仕様上、上記（ア）の法令名ではなく、当該法令名の法令の法令番号、例えば、「所得税法」の場合には「昭和四十年法律第三十三号」で指定する必要がある。しかし、不便であるため、上記（ア）に法令名を入力した後、下記（オ）の「OK」ボタンをクリックすることにより、検索サイトからの情報に基づき該当する法令番号を自動入力することができる。

(ウ)「基本条文数」テキストボックス

上記（イ）と同様に下記（オ）の「OK」ボタンをクリックすることにより、検索サイトから取得した情報に基づき当該法令の基本条文の総数（例えば、「所得税法」の場合、「243」件）が調査され、本テキストボックスに自動入力される。

(エ)「処理単位」テキストボックス

並列処理の際に、上記(ウ)の基本条文数を何条ごとに区分して並列処理するかを指定する。なお、並列処理される当該区分を「処理単位」という。例えば、処理単位が「1」である場合には、当該法令の全ての基本条文を並列的かつ個別に処理することとなる。

(オ)「ＯＫ」ボタン

上記（ア）及び（エ）を入力したのち、本ボタンをクリックすることにより、上述のとおり、検索サイトから取得した情報に基づき上記（イ）及び（ウ）が自動入力される。下記（カ）ないし（キ）の各ボタンをクリックして調査を開始する前に予めクリックしておく必要がある。

(カ)「開始（逐次）」ボタン

上記（イ）の法令番号の法令について「第一条」から順次、枝番も含めて次条を検索して、最終の条文（「所得税法」の場合、「第二百四十三条」）まで確認を行う。なお、最終の条文であるか否かは、当該条文の次条が参照可能か否かで判定する。

(キ)「開始（並列）」ボタン

上記（イ）の法令番号の法令について上記（エ）で指定されている処理単位で区分して、当該区分ごとに並列処理を行う。例えば、「昭和四十年法律第三十三号（所得税法）」の場合、基本条文の総数が 243 条あるので、処理単位を 15 条とした場合には、17 処理プロセスにより並列処理が行われる。

(ク)「開始（自動）」ボタン

上記（キ）の処理をいろいろな処理単位について自動的に実験するため、処理単位をまず基本条文の総数の半数から始めて調査したのち、当該処理単位を更にその半数として再調査を

行い、これを処理単位が１条となるまで自動的に繰り返す処理を行う。例えば、「所得税法」の場合、基本条文の総数243条であるため、まず、処理単位を「122」条、処理プロセス数を「２」重とし（このような処理方法を「[122／２]」と略記することとし、以下同様とする。）、さらに、[61／４]、[31／８]、[16／16]、[８／31]、[４／61]、[２／122]、[１／243] と計８回実験することとなる。

(ケ)「調査結果（逐次）」テキストボックス

本フォーム（図１　条文調査フォーム参照）中段のテキストボックスは、上記（カ）の調査結果を表示するためのテキストボックスである。当該調査結果の出力内容は、①法令名、②基本条文数、③各条文の処理が開始された時間（経過秒数）、④処理時間（経過秒数）、⑤基本条文１件当たり処理時間（秒数）、⑥確認済み条文数である。例えば、「所得税法」のある実行例の場合、それぞれ①「所得税法」、②243条、③０～1,683秒、④1,688秒、⑤6.95秒／件、⑥299条であった。

(コ)「調査結果（並列）」テキストボックス

本フォーム下段のテキストボックスは、上記（キ）又は（ク）の調査結果を共用して表示するためのテキストボックスである。当該調査結果の出力内容は、①法令名、②基本条文数、③処理単位、④処理プロセス数、⑤各プロセスで調査した各条文の処理が開始された時間（経過秒数）、⑥処理時間（経過秒数）、⑦基本条文１件当たり処理時間（秒数）である。例えば、「所得税法」のある実行例の場合、①「所得税法」、②243条、③31条、④８重、⑤０～243秒、⑥248秒、⑦1.02秒／件であった。ちなみに、上記（ケ）の例と比べると、並列処理化により基本条文１件当たりの処理時間が6.95秒／件から1.02秒／件に85%ほど減少していることが分かる。

3.2. 各ボタンの動作プログラム及び関連事項

本項では、上記3.1で紹介した各ボタンの動作プログラム及びその関連事項について、別添資料「条文調査フォームのソースコード」に基づいて概説する。なお、以下で例えば「[88-99]」とあるのは、当該ソースコードの行番号88行から99行までの部分を表す。

(ア)「ＯＫ」ボタン［17-30］

「OK」ボタンがクリックされると、イベント処理ハンドラ「LawNumberReady」が呼び出される［17-30］。そして、当該ハンドラでは、まず、上記3.1の（ア）に指定された法令名に基づいて検索サイトから該当する法令番号を検索する［19-19］。つぎに、当該法令番号に基づいて関数「GetNumberOfArticles」を用いて最後の基本条文番号（例えば、「所得税法」の場合「第二百四十三条」）を特定する［22-22］。なお、最後の基本条文番号については、法令取得APIを用いるのが本来であるものの、前述の制限があったため、条文内容取得APIを用いて二倍・二分探索法によって特定した。この二倍・二分探索法は、まず、①１条から始めて２条、４条、８条、16条…と２倍ずつ条文番号を増やしつつ条文内容取得APIを用いて条文検索を行い、基本条文が尽きて不成功（NotFound）になった基本条文番号を求め、つぎに、②その直前に成功（OK）した基本条文との間にあるはずの最後の基本条文番号を二分探索法で特定する方法である［80-99］。

その後、以上で特定できた基本条文の総数と上記3.1の（エ）で指定されている処理単位に基づき、いくつの処理プロセスを立ち上げる必要があるかを算出し変数「ParallelLevel」に格

納する［23-23］。

(イ)「開始（逐次)」ボタン［32-35］［101-121］

「開始（逐次)」ボタンがクリックされると、まず、イベント処理ハンドラ「StartSequential」が呼び出される［32-35］。そして、当該ハンドラでは、その処理をサブルーチン「GetAllArticleNumbers1」［101-121］に依頼している［33-33］が、当該サブルーチンでは、上記 3.1 のロで指定されている法令番号に基づき条文内容取得 API を用いて当該法令の特定の条文から次条を探索する関数「GetLatter」［158-191］を利用して、当該法令の１条から順次、次条を特定している［115-115］。なお、当該特定の方法は、上記「はじめに」で述べたような試行錯誤に基づく探索方法（以下「試行錯誤法」という。）に依っている。

(ウ)「開始（並列)」ボタン［37-44］［123-156］

「開始（並列)」ボタンがクリックされると、まず、イベント処理ハンドラ「StartParallel」が呼び出される［37-44］。当該ハンドラでは、その処理をサブルーチン「GetAllArticleNumbers2」［123-156］に依頼している［42-42］が、当該サブルーチンでは、変数「ParallelLevel」に設定されている件数の処理プロセスを立ち上げ［137-151］、各処理プロセスは、当該処理プロセスごとに変数「CurrentArticleNumber」へ設定される［141-141］基本条文番号から始めて、同様に変数「LimitArticleNumber」へ設定される［142-142］基本条文番号に到達するまで上記（イ）と同様な処理を行っている。

なお、上記 3.1 の（エ）の「処理単位」テキストボックスの値を適宜変更して実験することができるようにするため、同じ法令について当該テキストボックスの値が変更された場合には変数「ParallelLevel」の値を事前に再計算することとしている［38-41］。

おって、当該各処理プロセスにとってグローバルな変数については、配列の形で個別に用意する「12-15」とともに、初期設定している［126-131］。

(エ)「開始（自動)」ボタン［46-57］

「開始（自動)」ボタンがクリックされると、まず、イベント処理ハンドラ「StartParallelAuto」が呼び出される［46-57］。当該ハンドラでは、当該法令の基本条文を上記 3.1 の（ク）で述べたように処理単位を変更しながら［49-53］上記（ウ）と同様にサブルーチン「GetAllArticleNumbers2」に繰り返し依頼して［54-54］各処理を行っている。

(オ)　非同期のサーバー・アクセス処理について

上記各ボタンの処理に際しては、前述の各 API を用いたデータの遣り取りのために検索サイトと非同期でアクセスを行う次の三つのハンドラを使用している。すなわち、

① GetLawNumberAsync［59-78］…　法令名から法令番号を取得
② ServerAccessAsync1［203-227］…　法令番号と条文番号から条文内容を取得
③ ServerAccessAsync2［241-274］…　並列処理において上記②の処理を行うもの

なお、各ハンドラでは、サーバーとの HTTP 通信のために GetStringAsync メソッドを呼び出している［64-64］［213-213］［255-255］が、その際 ConfigureAwait メソッドに「False」を指定する(4)ことにより処理プロセスとのデッドロックを回避している。

(4) これによりサーバーから応答があった際に GetStringAsync メソッドを実行するために起動されたプロセスから待機中の各ハンドラにデータが引き継げるようになる。なお、Await による処理プロセス待機処理の際のデッドロックの発生メカニズム等の詳細については、例えば、ufcpp「async/await と同時実行制御」WordPress.com Blog（2012

4. 実験結果

4.1. 調査対象法令

今回の実験では、上記「はじめに」で述べたように本件の並列処理が必要となった原因である、法令取得 API の対象外であった①「所得税法」、②「租税特別措置法」、そして、それらとの比較のために、③条文数がより少ない「消費税法」、そして、④基本条文数が最大の「民法」を取り上げることとした。

4.2. ターンアラウンド・タイム

検索サイトとのアクセスに要する平均的なターンアラウンド・タイムを調査対象の法令ごとに計測した結果は、表1のとおりである。

表 1　各法律のターンアラウンド・タイム一覧

（件・秒／件）

項目 / 法令	基本条文数	成功	不成功	成功不成功倍率
① 所得税法	243	0.75	4.06	5.4
② 租税特別措置法	98	1.15	2.65	2.3
③ 消費税法	67	0.37	0.45	1.2
④ 民法	1,050	0.29	0.69	2.4
単純平均値	365	0.64	1.96	2.8

なお、成功（OK）の場合の計測値は、各法令の全ての基本条文をアクセス対象として計測したものの平均値であり、他方、不成功（NOT FOUND）の場合の計測値は、各法令の全ての基本条文について当該条文番号に存在し得ない枝番「の一」を付した条文番号（例えば、「第二条の一」）についてアクセス対象として計測したものの平均値である。ただし、実験の実施時期によって計測値にバラツキが生じた（例えば、通信環境の影響か、処理効率については一般に深夜の方が昼間よりも高い傾向がある。）ため、各法令について最終的な実験をそれぞれ２回ずつ行い、より少ない方の計測値を採用した。したがって、必ずしも再現性のある安定的な結果ではないため、飽くまでも参考値として理解されたい。

ただし、取り敢えず表１の結果を評価すれば、法令によって各項目とも大きく測定結果が異なり、特に所得税法では、成功の場合のターンアラウンド・タイムが 0.75 秒／件であるのに対して、不成功の場合のターンアラウンド・タイムが 4.06 秒／件と調査対象法令の中では最大であり、成功に対する不成功の倍率（以下「成功不成功倍率」という。）も 5.4 倍と最大で

年）（令和２年４月３日現在）https://ufcpp.wordpress.com/2012/11/12/asyncawait%E3%81%A8%E5%90%8C%E6%99%82%E5%AE%9F%E8%A1%8C%E5%88%B6%E5%BE%A1/ など参照。

ある。また、最小のターンアラウンド・タイムは、成功の場合で民法の 0.29 秒／件であり、不成功の場合で消費税法の 0.45 秒／件であった。また、最小の成功不成功倍率は、消費税法の 1.2 倍であった。

したがって、サーバーにおける各法令の処理に、そのようなバラツキがあること自体は明らかといえよう。そして、そのようなバラツキの理由ないし原因については、定かではないものの、少なくとも所得税法の成功不成功倍率が大きいことは、結果として、前述した試行錯誤法の効率を低下させる主な原因となっているものと考えられる。ただし、その反面で、以下に述べるように並列処理の効果が大きいであろうことを示唆しているともいえよう。

4.3. 逐次処理結果

検索サイトとの個々のアクセスの完了まで待機しながら逐次処理により試行錯誤法を実行して、調査対象の各法令について基本条文１件当たりの平均的な処理時間を計測した結果は、表 2 のとおりである。

表 2　逐次処理結果一覧

（件・%・秒／件）

項目 / 法令	基本条文数	確認済条文数	枝番割合	デスクトップ	ノート
① 所得税法	243	299	18.7	6.95	7.14
② 租税特別措置法	98	577	83.0	31.51	31.68
③ 消費税法	67	76	10.5	1.66	1.67
④ 民法	1,050	1,201	12.6	2.26	2.22

なお、ここで「枝番割合」とは、結果的に試行錯誤法により確認できた条文数（以下「確認済条文数」という。）のうち枝番付の条文の数の割合をいう（以下同じ。）。また、上記 4.2 と同様に、各法令について最終的な実験をそれぞれ２回ずつ行い、より少ない方の計測値を採用した。

4.4. 並列処理結果

複数の処理プロセスによって試行錯誤法を並列的に実行して、ある処理プロセスが待機中にも検索サイトからの回答を受け取った他の処理プロセスについては、その実行を可能として総合的に検索サイトとのアクセスの効率化を図った上で、全体的な処理効率を把握するために調査対象の各法令について全体としての処理時間を計測した結果は、表 3 のとおりである。

表 3　並列処理結果一覧

（件・秒）

項目 / 法令	処理単位	処理プロセス数	デスクトップ	ノート
① 所得税法 ・基本条文数 243 件 ・確認済条文数 299 件	243	1	1,688	1,736
	122	2	927	815
	61	4	463	476
	31	8	248	221
	16	16	157	141
	8	31	108	106
	4	61	108	99
	2	122	89	103
	1	243	94	99
② 租税特別措置法 ・基本条文数 98 件 ・確認済条文数 577 件	98	1	3,088	3,105
	49	2	2,101	2,083
	25	4	1,555	1,495
	13	8	1,283	1,286
	7	14	1,182	1,186
	4	25	854	847
	2	49	680	697
	1	98	593	583
③消費税法 ・基本条文数 67 件 ・確認済条文数 76 件	67	1	111	112
	34	2	56	58
	17	4	32	30
	9	8	18	21
	5	14	11	10
	3	23	8	7
	2	34	9	7
	1	67	5	7
④民法 ・基本条文数 1,050 件 ・確認済条文数 1,201 件	1,050	1	2,378	2,334
	525	2	1,254	1,288
	263	4	664	662
	132	8	414	416
	66	16	216	221
	33	32	152	160
	17	62	83	96
	9	117	79	93
	5	210	87	90
	3	350	90	97
	2	525	103	107
	1	1,050	117	132

なお、処理プロセス数を徐々に増やしながら計測したが、その数については、処理単位を半減しつつ最終的に１件となるように定めることとした（上記 3.1 の（ク）参照)。また、上記 4.3 と同様に、各法令について最終的な実験をそれぞれ２回ずつ行い、より少ない方の計測値を採用した。

そして、表 3 の結果からいえば、デスクトップの方がノートよりも若干処理時間が少ないような傾向がみられるものの、ほぼ無視できるといえよう。結局、上記 4.3 と同様に、計測結果は、使用したパソコンの処理能力にはあまり影響を受けなかったといえよう。

また、各法令とも、処理プロセス数の増加により並列度が高まるにつれて、民法以外では、全体的な処理時間がほぼ単調に減少している。ただし、民法では、処理単位が５件近辺に最適値があるようである。そこで、各法令について処理プロセス数が１（つまり、逐次処理）の場合の処理時間に対する、並列処理における最低の処理時間の比率をデスクトップの場合でみれば次のとおりである。

① 所得税法　　89 秒／1,688 秒≒5.3%
② 租税特別措置法　593 秒／3,088 秒≒19.2%
③ 消費税法　　5 秒／111 秒≒4.5%
④ 民法　　79 秒／2,378 秒≒3.3%

上記のとおり、今回の実験では、租税特別措置法を除き、並列化により処理時間を逐次処理のほぼ 20 分の１以下にすることが可能であったといえよう。なお、租税特別措置法において５分の１ほどと時短効果が少ないのは、やはり枝番割合が高いためであろう。

5. e-Gov 法令検索サイト用 API の改善提案

やはり、当初一部とはいえ、所得税法などの重要な法令の全体像が API で把握できなかったことは、法令改正に柔軟かつ迅速な対応が求められるタイプのアプリケーション開発の場合には大きな障害となっていたであろう。しかし、たとえそのような状況であったとしても、個々の条文をアクセスする場合に、少なくとも当該条文の直前及び直後の条文番号が容易に把握できれば、そのデメリットが緩和できるものと考えられる。

また、今回試みたような試行錯誤法などの便宜的な対処方法の精度は、ネットワークの通信環境の品質に大きく依存するものと考えられる。実際、今回の実験でも、各処理プロセスにおける通信エラー発生時における処理を簡略化した（成功「OK」以外は、その他の通信エラーも含めて一律に不成功（NOT FOUND）とみなした）ため、民法に係る並列度の非常に高い実験において処理結果が一部欠けてしまう場合があったが、いずれにせよ、今回のように本来逐次的な処理を並列度の高い並列処理によって実行する場合においては、極少数の処理プロセスが何らかの理由でダウンしただけで他の処理プロセスにおける正常な処理が結局無駄になってしまう恐れが高いであろう。

そこで、特定の主要な法令を法令取得 API の対象外とする理由がタイムアウト発生の危険性を回避するためだけであれば、その対処策として、例えば、①法令取得 API の仕様を変更して、サイズの大きな法令については、条文のテキスト部分を省略した XML データを返すように修正してはどうであろうか。つまり、条文本体は、必要に応じて条文内容取得 API で間接的に取得するようにするわけである。あるいは、②条文内容取得 API の仕様を変更して、当該条

文の直前及び直後の条文番号を返すように当該 API を修正してはどうであろうか。これにより、たとえ法令の全体が把握できなかったとしても、特定の条文を基準として画面を上下にスクロールするような処理が容易となろう。

6. おわりに

パソコンの処理能力の向上は、もはや「絶大」としかいいようのないレベルになったといえるであろう。40 数年前には、例えば、メインフレームでも 1 台の CPU で限られた主メモリー（例えば、当方の母校の場合 192KB）をどのように有効かつ柔軟に複数のユーザーで共有するかが主なテーマであったものが、今回使用したデスクトップでさえ、6 CPU で 16GB を一人で専有できる時代になった。至極単純に比較すれば、能力的には（6 CPU×16GB）／（1 CPU×192KB）≒500,000 倍であり、そのほか、動作速度も相当に早くなっている（マイクロプロセッサの動作周波数でみれば、750KHz（1971 年）[5]から 3.2GHz（当デスクトップ）と 4,000 倍以上になっている。）ので、それを含めれば、なんと 500,000 倍×4,000 倍＝20 億倍である。

それに比べて、使う側である我々人間サイドの能力は、それほど向上してはいないため、上記のようなコンピューターの能力を十分に活用するためには、今後、更にコンピューターの手助けを受けながら我々人類も研鑽していく必要があろう。そして、多分、プログラムの並列化の分野もその一つといえよう。本稿がそのような観点からも読者の何らかの参考になれば幸いである。

参考文献

以下に本稿では個別には引用していないものの全般的に参考にした文献を掲げる：

[1] ナルボ『Visual Studio パーフェクトガイド』（技術評論社・2019 年）

[2] Microsoft「チュートリアル：Async と Await を使用した Web へのアクセス（Visual Basic）」（令和 2 年 4 月 6 日現在） https://docs.microsoft.com/ja-jp/dotnet/opbuildpdf/visual-basic/programming-guide/concepts/async/toc.pdf?branch=live 参照

[3] こだかかおる「ファイルとフォルダ名の取得を通じてマルチスレッド化の手法を学ぶ」『VB paradise』日経ソフトウェア 2008.4、121 頁～129 頁

[4] 原田英生ほか「特集２マルチコアを使いこなせ！並列プログラミング入門」日経ソフトウェア 2010.9、50 頁～73 頁

[5] 佐野正博「Intel 社が開発したマイクロプロセッサーの技術的スペックの歴史的変遷」（令和２年４月６日現在）https://www.sanosemi.com/history_of_Intel_CPU_techspecs-mini.htm 参照。

【追記】

2020年11月におけるe-Gov法令検索サイトのリニューアルについて

本稿脱稿後、2020年（令和２年）11月24日付でe-Gov法令検索サイト（https://elaws.e-gov.go.jp）が大幅にリニューアルされた。これまでなかった更新対象法令を把握するためのAPI（更新法令一覧取得API）が新設されるとともに、本稿で前提となっていた弱点、すなわち、法令取得APIにおいて所得税法等のデータサイズの大きい法令についてAPIによる取得対象外とされていた点についても改良され、各法令の全文データを一括して随時に取得することが可能になった。そのため、本稿で紹介したような前後の条文をダイナミックに探索するような手法をあえて採らずとも、必要に応じて随時当該法令の全文データを取得した上で、各条文の前後関係を一括して把握することが容易となった。そこで、本稿で検討対象とした22法令について、その全ての条文のXMLデータを取得し、それに基づいて前後の条文を特定する方法の実用性について実際に検討してみることとした。

その結果、22法令の全てについて各条文データ（22法令計4,923件）の前条・次条をプログラムにより自動的に分析・把握してみたところ、全条文データのダウンロードに要する通信時間も含めて、高々１～２分程度で詳細に分析することが可能であり、ほぼリアルタイムに把握できることが確認できた。したがって、その結果や法令改正の頻度等も踏まえれば、本稿で紹介したような前後の条文をダイナミックに探索する方法をあえて採用するよりも、むしろ現状では、事前に各条文の前後関係の情報を一括して把握した上で、その結果に基づいて前後の条文を特定することを基本として、適宜のタイミングで必要に応じて当該情報をアップデートする方法の妥当性がより高まったと考えられるので、その旨申し添えることとしたい。

（別添資料）条文調査フォームのソースコード