# 並行XMLパーサの提案

蜂巣吉成
南山大学数理情報学部

**概 要**

本稿では XML パーサを，前処理，実体化処理，検証処理の 3 つの処理に分け，それぞれを並行実行する手法を提案する．並行 XML パーサは 2 つの生産者/消費者モデルとして形式化することができる．並行 XML パーサを試作し，実験によりその効果を見積もった．

## 1 はじめに

近年の XML の普及に伴い XML 文書が大規模化しており，XML パーサの処理がアプリケーションの処理時間の大部分を占めるという問題が指摘されている [1]．一般に，XML パーサは XML 文書を木構造データに変換し，スキーマに合致しているか検証を行う．アプリケーションは XML パーサにより作成された木を利用して処理を行う．

XML パーサの高速化を行うために，われわれは既に並行 XML パーサを提案した [2]．並行 XML パーサは木作成処理を前処理と実体化処理の 2 つに分け，それらを並行実行する．前処理は XML 文書を解析して内部データとして保存する．実体化処理は内部データから木の節点を作成する．

本稿では並行 XML パーサを拡張し，前処理と実体化処理に加えて，検証処理も並行実行する XML パーサを提案する．われわれは並行 XML パーサを 2 つの生産者/消費者モデル，すなわち，前処理が生産者で実体化処理が消費者のモデルと実体化処理が生産者で検証処理が消費者のモデルの組み合せとして形式化した．並行 XML パーサを試作し，1 つの CPU を持つ計算機で実験を行い，その効果を見積もった．

なお，以降，前処理と実体化処理を並行実行する XML パーサを単純並行 XML パーサ，前処理，実体化処理と検証処理を並行実行する XML パーサを並行 XML パーサと呼ぶ．

## 2 関連研究

遅延 XML パーサ [3] と単純並行 XML パーサ [2] について述べる．単純並行 XML パーサは遅延 XML パーサのアイディアを発展させたものである．

### 2.1 遅延 XML パーサ

遅延 XML パーサは前処理と実体化処理の 2 種類の処理で構成され，アプリケーションから実際に参照される木のみを作成する (図 1)．

前処理は XML 文書を読み込み，木の節点毎にその種類 (要素やテキストなど)，親子兄弟関係，テキスト表現 (要素名やテキストなど) などを抽出して内部データとして保存する．前処理終了後，パーサは木の根となる節点のみを実際に作成する．そのほかの節点は仮想節点である．アプリケー

ションが仮想節点を参照しようとした場合，それをトリガにして実体化処理が実行される．実体化処理はオンデマンドで内部データから実際に節点を作成する．

遅延 XML パーサではアプリケーションからは参照されない木の節点の作成処理を省くことで，従来の XML パーサより高速に処理が行える．しかし，アプリケーションが木のほとんどの部分を参照する場合，そのオーバヘッドのために従来の XML パーサよりも処理時間が増大してしまう．

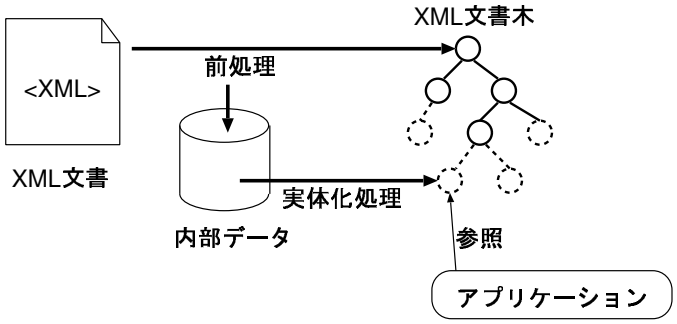


図 1: 遅延 XML パーサの概念図

### 2.2 単純並行 XML パーサ

われわれは既に XML パーサの処理を機能分割とデータ分割の観点から整理し，単純並行 XML パーサを提案した．機能分割では XML パーサを並行実行可能な複数のプロセスに分割し，データ分割では XML パーサが処理するデータを並行処理可能な複数のデータに分割する．

われわれは遅延 XML パーサの前処理と実体化処理を機能分割の結果と考え，生産者/消費者モデルとして形式化した．

- 前処理と実体化処理は内部データを共有する．内部データは共有バッファに相当する．
- 前処理は実体化処理で利用されるデータを生成するので，生産者に相当する．
- 実体化処理は前処理が生成したデータを利用するので，消費者に相当する．

データ分割の観点からは，内部データが分割可能である．XML 文書は最初から最後まで逐次的に解析する必要があり，分割することはできない．したがって，前処理プロセスは 1 つである．一方，内部データとして保存されている節点の情報は他の節点の情報とは独立しているので，内部データを分割して複数の実体化処理プロセスで並行に処理することができる．

2 つの実体化処理プロセスをもつ単純並行 XML パーサの概念図を図 2 に示す．小さな四角が内部データであり，影付きの四角がまだ実体化処理されていない節点の情報である．

## 3 並行 XML パーサ

一般に XML パーサは XML 文書から木を作成し，さらに XML 文書がスキーマに合致しているか検証を行う[1]．本稿では，単純並行 XML パーサを拡張し，検証処理も並行実行する並行 XML パーサを提案する．

[1]SAX パーサのように木を作成しないパーサも存在するが，アプリケーションが XML 文書をランダムにアクセスする場合はパーサに木を作成させることが多い．

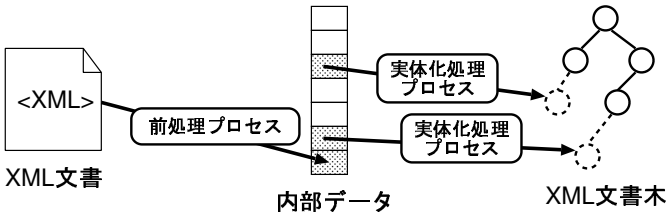


図 2: 単純並行 XML パーサの概念図

## 3.1 モデル

機能分割とデータ分割の観点から並行 XML パーサのモデルを考える．

並行 XML パーサは，前処理，実体化処理，検証処理の 3 つに機能分割できる．検証処理プロセスは実体化処理プロセスによって生成された木が，スキーマに合致しているか調べる．木を共有バッファとみなすことで，実体化処理と検証処理を生産者/消費者モデルと考えることができる．

データ分割の観点から，木を重複のない複数の部分木に分割し，複数の検証処理プロセスを用いることを考えた．しかし検証処理では，複数の節点に対してその間の関係を調べる必要がある．節点が相互に依存しており，木を独立した部分木に分けることは困難である．よって，木は分割せずに検証処理プロセスは 1 つとする．

以上より並行 XML パーサは，1 つの前処理プロセス，複数の実体化処理プロセスと 1 つの検証処理プロセスから構成され，2 つの生産者/消費者モデルを組み合わせたモデルと考えることができる．図 3 に並行 XML パーサの概念図を示す．

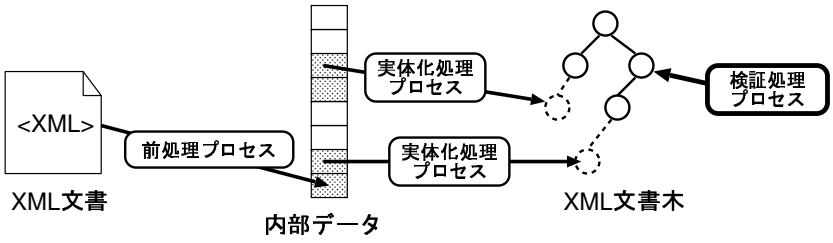


図 3: 並行 XML パーサの概念図

## 3.2 実現

並行 XML パーサを試作したので，その実現方法を示す．高速な処理が必要とされるアプリケーションは C 言語で実現されることが多いと考えて，実現には C 言語と pthread ライブラリを用いた．試作したパーサは，前処理プロセス，実体化処理プロセス，検証処理プロセスをそれぞれスレッドとして実行し，C 言語における XML パーサの一つである libxml2 [4] と互換性のある木を作成する．パーサの規模は C，flex，yacc で 5,000 行程度である．

### 3.2.1 前処理

前処理プロセスは XML 文書を読み込み，トークン列に分解し，トークンの種類やトークン間の関係などを解析し，それらを内部データとして保存する．トークンは<，</，> などで区切られた

文字列である．トークンには出現した順序で番号がつけられ，この番号を用いてトークン間の親子兄弟関係を表す．

並行 XML パーサの処理速度は前処理プロセスの処理速度に依存する．つまり，前処理プロセスが遅い場合，実体化処理プロセスが飢餓状態となり，全体の速度低下につながる．前処理の高速化のために，属性は前処理プロセスでは解析せずに，実体化処理プロセスで解析する．

前処理はコンパイラにおける字句解析に類似しているので，その実現には字句解析プログラム flex を用いた．トークンをパターンとして，解析処理をそのアクションとして記述した．

#### 3.2.2 内部データ

文献 [3] を基にして内部データを設計した．内部データは各節点データの配列で表現される．節点データは節点の種類，親子兄弟関係にある節点のトークン番号，節点が表すテキスト表現へのポインタなどから構成される．配列には前処理プロセスが作成したデータの最後を表すインデックス (前処理インデックス) と，実体化処理プロセスが処理したデータの最後を表すインデックス (実体化処理インデックス) がある．前処理インデックスと実体化処理インデックスの値が同じ場合，実体化処理プロセスは待機状態となる．

#### 3.2.3 実体化処理

実体化処理プロセスは内部データから，libxml2 と互換性のある XML 文書の木を作成する．属性の解析も実体化処理で行われる．

複数の実体化処理プロセスを用いて並行処理を行う場合は，実体化処理プロセスが内部データの実体化処理インデックスに対して同期をとり，処理を行う．

#### 3.2.4 検証処理

検証処理プロセスは木に対して，DTD に合致しているか検証を行う．DTD は XML Schema[5] などに比べて構造が単純で検証処理の実現が容易であり，本手法の有効性を早期に確認するのに適していると考えた．

検証処理プロセスは DTD ファイルを解析し，妥当な XML 文書の構造を表現した木オートマトンと，属性名と属性値の型などからなる属性表を作成する．その後，それらを用いて，実体化処理が作成した木に対して検証を行う．

DTD ファイルの解析と木オートマトンの構築は，プログラミング言語の構文解析処理と構文木作成処理に類似しているので，その実現には flex と yacc を用いた．

## 4 実験

試作した並行 XML パーサの効果を見積もるために 1 つの CPU を持つ計算機上で実験を行った．計算機は 512MB のメモリを搭載した Pentium M 1.7GHz で，OS には Linux 2.4.22 を用いた．実験で用いた XML 文書は XML Benchmark[6] プログラムで生成した (表 1)．ファイルには要素数が多いもの (A) と，属性とテキスト中の文字数が多いもの (B) の 2 種類を用意した．

表 2 に各処理に要した時間を示す．実験より，以下のことが分かった．

表 1: XML 文書

| ファイル | サイズ | 要素数 | 属性数 | テキスト節点数 | 文字数 |
|---|---|---|---|---|---|
| (A) | 25MB | 1,161,980 | 16,093 | 93,6021 | 7,527,555 |
| (B) | 50MB | 64,2371 | 1,052,422 | 1,669,403 | 26,989,235 |

文字数はテキスト節点数に含まれる文字数である．

表 2: 実行時間

| ファイル | 前処理 | 実体化処理 | 検証処理 | 合計 |
|---|---|---|---|---|
| (A) | 992(43.1%) | 443(19.2%) | 742(32.2%) | 2300 |
| (B) | 1532(48.3%) | 738(23.2%) | 807(25.4%) | 3170 |

(単位: ミリ秒)

- 複数の CPU を持つ計算機で各処理がオーバラップして実行されれば，全体で 2 倍程度の高速化が望める．
- ファイル (B) は属性の数が多いので，ファイル (A) よりも実体化処理の占める割合が高くなっていると考えられる．
- 前処理プロセスが全体の処理の半分程度を占めており，高速化における律速段階となる．並行 XML パーサ全体の処理速度を向上させるには前処理の高速化が必要である．
- 実体化処理と検証処理の処理時間の合計は前処理の処理時間と同程度なので，これらをまとめて一つのスレッドで実行すればオーバヘッドの削減が望める．

# 5 おわりに

本稿では並行 XML パーサを提案した．並行 XML パーサは前処理，実体化処理，検証処理で構成され，生産者/消費者モデルとしてモデル化できる．並行 XML パーサを試作して実現方法を示し，実験によりその効果を見積もった．

以下を今後の課題として挙げる．

- 複数の CPU を持つ計算機上で評価を行う．
- 高速な前処理プロセスを設計・実現する．今回は汎用の字句解析生成プログラム flex を用いて前処理プロセスを実現したが，XML 文書処理に特化したプログラムを作成することで，処理速度の高速化が望めると考えられる．
- XML Schema[5] や RELAX NG[7] などのスキーマ言語に対する検証処理プロセスを実現する．これらの言語は DTD よりも検証処理に時間を要すると思われるので，並行処理の効果は大きいと考えられる．

## 謝辞

本研究は 2006 年度南山大学パッヘ研究奨励金 I-A-2 (Nanzan University Pache Research Subsidy I-A-2 for the 2006 academic year) の助成を受けた．

# 参考文献


[1] Matthias Nicola and Jasmi John. XML parsing: a threat to database performance. In *Proceedings of the 12th International Conference on Information and Knowledge Management*, pp. 175–178. ACM Press, 2003.

[2] Yoshinari Hachisu. Towards XML Concurrent Parsing. In *IADIS Virtual Multi Conference on Computer Science and Information Systems (MCCSIS2005)*, p. 5, Apr 2005.

[3] Markus L. Noga, Steffen Schott, and Welf Lowe. Lazy xml processing. In *Proceedings of the 2002 ACM symposium on Document engineering*, pp. 88–94. ACM Press, 2002.

[4] Daniel Veillard. *The XML C library for Gnome*. http://xmlsoft.org/.

[5] World Wide Web Consortium. *XML Schema*. http://www.w3.org/XML/Schema.

[6] *XML Benchmark*. http://xmlbench.sourceforge.net/.

[7] James Clark and Makoto Murata. *RELAX NG*, 2001. http://relaxng.org.