ІФОРМАЦІЙНІ ТЕХНОЛОГІЇ

doi: https://doi.org/10.15407/emodel.42.01.033
УДК 004.383

А.М. Сергієнко, д-р техн. наук,
М.М. Орлова, канд. техн. наук, **О.А. Молчанов**, аспірант
Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
(Україна, 03065, Київ, пр-т Перемоги, 37,
e-mail: aser@comsys.kpi.ua, orlova@ua.fm,
oleksii.molchanov@gmail.com)

Апаратно-програмна обробка XML-документів

Розглянуто існуючі алгоритми та засоби обробки XML-документів. Встановлено необхідність використання високопродуктивних пристроїв аналізу XML-запитів, які швидко переналаштовуються на інші граматики. Розроблено процесорне ядро SM16, яке ефективно виконує стекові алгоритми парсингу XML-документів і реалізується у програмованих логічних інтегральних схемах (ПЛІС). Процесорне ядро має архітектуру стекового процесора, до якого додано три блоки стекової пам'яті, хеш-таблиця та команди, що прискорюють виконання операцій парсингу. Запропоновано апаратно-програмну систему на основі ПЛІС, яка має ведучий процесор та від десятків до сотень виконавчих процесорних елементів SM16. Ця система не тільки ефективно обробляє XML-документи, але й швидко переналаштовується на обробку документів з іншими граматиками.

К л ю ч о в і с л о в а: XML, парсер, стековий процесор, граматика, стековий автомат.

Широке впровадження мови XML у веб-службах обумовлено її стандартизацією і притаманною властивістю розширюваності. Наразі існують тисячі програмних застосувань, що можуть обробляти дані у форматі XML. При цьому розмір документів XML налічує від сотень байт до десятків гігабайт [1]. Найбільше XML-документів обробляється у формі запитів до веб-серверів. Останнім часом XML використовується для обміну даними в Інтернеті речей, для чого розроблено спеціальний профіль DPWS (Devices Profile for Web Services) [2]. Для збільшення пропускної здатності каналів передачі введено новий формат XML — Efficient XML Interchange (EXI) [3].

Незважаючи на те що XML має багато переваг, до числа яких входять деталізація і описовий характер, синтаксичний аналіз XML-запитів

спричиняє значне гальмування продуктивності у веб-службах [1]. Розбір XML-документів займає багато пам'яті та обчислювальних ресурсів, споживаючи приблизно 30% часу обробки в застосуваннях веб-служб [4]. Досвід використання XML з базами даних засвідчує, що розбір XML є основним «вузьким місцем» у нарощуванні продуктивності і може збільшити вартість транзакцій більше, ніж у 10 разів [5].

Розглянемо алгоритми розбору XML-документів та засоби його прискорення. Пропонується виконувати такий розбір за допомогою апаратно-програмних засобів, реалізованих у програмованих логічних інтегральних схемах (ПЛІС).

Алгоритми розбору XML-документів. Поширені алгоритми розбору XML-документів, тобто парсери, бувають двох категорій: парсери, керовані подіями, і парсери, основані на побудові дерева розбору. Парсер, керований подіями, який встановлено у сервері, спочатку аналізує чергову XML-фразу, а потім через зворотний зв’язок повідомляє клієнтське застосування про черговий тег, який він знайшов. За цією подією це застосування передає для аналізу наступну XML-фразу. Як наслідок, парсер, керований подіями, не завантажує увесь XML-документ за один раз і тому потребує невеликого обсягу пам'яті. Прикладом такого парсера є синтаксичний аналізатор SAX, який вважається галузевим стандартом [6]. У більшості випадків задача парсера полягає у підтвердженні правильності документа, тобто його відповідності конкретній граматиці та читанні його окремих полів. Тоді такий парсер виконує задачу фільтрації документів.

Парсер, оснований на побудові дерева, записує весь вміст XML-документа в оперативну пам'ять сервера і створює об'єктну модель документа (DOM) у вигляді дерева, яке його відображає [7]. DOM забезпечує максимальну гнучкість для користувачів, але за це необхідно платити великими витратами на об’єм пам’яті та жорсткими вимогами до обчислювальної системи.

Для задання структури XML-документа у парсерах використовують ту чи іншу мову схем XML, які зазвичай формалізуються за допомогою регулярної граматики дерев [8, 9]. Граматика конкретного виду запитів у вигляді профіля подається за допомогою мов запитів XML, таких як XPath [10]. У загальному випадку така граматика задається як четвірка $G = (N, T, S, P)$. Тут N — скінченна множина нетермінальних символів; T — скінченна множина термінальних символів; $S \subseteq N$ — множина початкових символів; P — сукупність породжуючих правил виду $X \to ar$, де $X \in N$, $a \in T$; r — регулярний вираз над N. Це правило свідчить про те, що X породжує піддерева з коренем a та дітей, які задовольняють вираз r [9].

Розглянемо приклад такої граматики:

#Grammar
N = { Data, Header, Body, Content }
T = { d, h, b, c } # теги для кожного імені тегу
S = { Data }
P = # Правила, описані нижче
#Rules:
Data → '<d>' Header Body* '</d>' ; # Rule1
Header → '<h>' '</h>' ; # Rule2
Body → '' '' ; # Rule3
Body → '' Content '' ; # Rule4
Content → '<c>' '</c>' ; # Rule5

Наступний приклад XML-документа відповідає даній граматиці:
Приклад XML:

<d> <h></h> <c></c>   </d>. (1)

При розборі цього документа повинно бути побудовано дерево таке, як зображено на рис. 1. Кожна вершина графа відповідає певному вузлу документа. Сусідні вершини, які розміщені вище, є батьківськими вузлами, а ті, що лежать нижче, — дочірніми.

Розглянемо простий алгоритм перевірки, чи документ дійсний щодо заданої регулярної граматики дерев [9]. Алгоритм реалізований у стековому скінченному автоматі (СА), який має три стеки: P, Y та S. Він виконує обхід дерева, зображеного на рис.1, вглиб та спрацьовує за подіями, а саме фразами документа, які трапляються у тому порядку, в якому вони слідують при його читанні. Серед них є відкриваючий *e* (<…) і закриваючий $\bar{e}$ (</…) теги.

Алгоритм використовує три стеки наступним чином:

стек P зберігає набори символів з *N*; його вершина вказує символи з *N*, які можна використовувати для узгодження наступного вузла в документі; ініціалізується множиною *S*;

стек Y зберігає набори правил виводу з *P*; кожен набір вказує кандидатів з набору правил, які можуть відповідати поточному вузлу в документі; під час запуску алгоритма стек Y є порожнім;

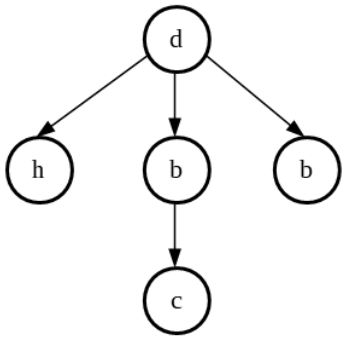

Рис. 1. Приклад графа XML-документа

стек S зберігає списки наборів символів з N, де перший набір символів позначає символи з N, які відповідають першому дочірньому поточному вузлу (визначається у момент опрацювання алгоритмом кінцевого елемента події для цього дочірнього вузла), другий набір символів позначає символи з N, які відповідають другому дочірньому поточному вузлу (визначається у момент опрацювання алгоритмом події кінцевого елемента для цього вузла) тощо.

А л г о р и т м, реалізований у стековому СА:

початковий стан: на вхід подається XML-документ D, стеки P, Y та S — пусті;

додати в стек P набір початкових нетермінальних символів;
обійти D вглиб {
якщо зустрівся тег e {
обираються правила виду $X_i \to a\ (r_i)$ такі, де a — ім'я тегу e,
$X_i \in N$'— один із символів у списку у вершині стеку P;
якщо (N' = ∅) {повідомлення, що D — некоректний; кінець;}
список вибраних правил додається до Y;
додається пустий список в стек S;
додається список нетермінальних символів з r_i в стек P;
}
якщо зустрівся тег $\bar{e}$ {
виштовхується список правил $\{X_i \to a\ (r_i)\}$ зі стеку Y;
виштовхується список нетермінальних символів $\{X_i\}$ зі стеку S;
обирається множина нетермінальних символів $X' \in \{X_i\}$;
якщо (X'= ∅) {повідомлення, що D — некоректний; кінець;}
X' додається до стеку S;
виштовхується список нетермінальних символів зі стеку P;
}
}
повідомлення, що D — коректний;
кінець.

Процес виконання цього алгоритму при розборі XML-документа (1) відображено у табл. 1. У роботі [9] описано й інші алгоритми розбору. Даний алгоритм та приклад розбору є спрощеними. Але вони показують основні властивості, притаманні алгоритмам розбору XML-документів, зокрема те, що для реалізації парсера, керованого подіями, доцільно використовувати автомат, який має три стеки. Ефективність виконання розбору залежить від способів реалізації цього автомату, які розглянемо далі.

Прискорення фільтрації XML-документів. Фільтрація XML-текстів є складною проблемою, бо вона повинна підтримувати вчасну оброб-

ку великих об'ємів вхідних потоків різноманітних XML-запитів до вебсерверів. Для підвищення продуктивності обробки XML-запитів запропоновано багато методів та засобів прискорення. Розглянемо деякі з них.

Програмні засоби прискорення. В [11] запропоновано перекодувати термінальні та нетермінальні символи у бінарні коди для зменшення об'єму пам'яті та збільшення швидкості доступу до даних. Програма попереднього сканування [12] будує дерево документа XML для подальшого його розбиття і паралельної обробки. В [13] використано паралелізм з розділенням процесу розбору XML на кілька фаз для подальшого планування обробки потоків запитів в конвеєрній моделі.

У роботах [14,15] запропоновано побудувати XML-фільтр (XFilter) як СА, реалізований програмно. Використано об'єднання таких СА для

Таблиця 1. **Розбір XML-документа (1)**

Крок	Вхідний тег	Стек P	Стек Y	Стек S
1		(Data)		
2	<d>	(Header, Body) (Data)	(Rule1)	()
3	<h>	(Header, Body) (Data)	(Rule2) (Rule1)	() ()
4	</h>	(Header, Body) (Data)	(Rule1)	((Header))
5		(Content) (Header, Body) (Data)	(Rule3, Rule4) (Rule1)	() ((Header))
6	<c>	() (Content) (Header, Body) (Data)	(Rule5) (Rule3, Rule4) (Rule1)	() () ((Header))
7	</c>	(Content) (Header, Body) (Data)	(Rule3, Rule4) (Rule1)	((Content)) ((Header))
8		(Header, Body) (Data)	(Rule1)	((Header, Body))
9		(Content) (Header, Body) (Data)	(Rule3, Rule4) (Rule1)	() (Header), (Body)
10		(Header, Body) (Data)	(Rule1)	(Header), (Body), (Body)
11	</d>	(Data)		

кількох різних запитів XML. Згідно з підходом LazyDFA у процесі фільтрації динамічно будується слабодетермінований СА [16]. У роботі [17] використано динамічний автомат для пошуку співпадіння за алгоритмом Ахо—Корасіка. Такий автомат дає змогу шукати співпадіння між XML-запитом та запитом з деякої множини очікуваних запитів і таким чином зменшити витрати на реалізацію автомату.

Апаратні засоби фільтрації з фіксованим алгоритмом. Для апаратної фільтрації запитів XML ефективним є використання ПЛІС, у якій зконфігуровано СА, налаштований на певний набір граматик [18]. У роботі [19] прискорення фільтрації одержано в результаті використання асоціативної пам'яті для збереження дерева документа. У роботах [20—22] описано системи на основі СА зі стеками, побудованими за допомогою компіляції граматик XML-документів у апаратний автомат.

Як свідчать результати досліджень, парсер, реалізований апаратно на основі ОС, має найбільшу швидкодію, але він налаштований на конкретний XML-запит або обмежену кількість запитів, що обробляються. При додаванні до списку запитів хоча б одного нового запиту проект слід перекомпілювати, що вимагає великих часових витрат.

Апаратні засоби фільтрації зі змінним алгоритмом. Для прискорення модифікації системи фільтрації запитів у роботі [23] запропоновано кістяковий СА, спроможний до переналаштування на обробку нових запитів. Для цього проект, конфігурований в ПЛІС, виконано як заготовка СА з невизначеними умовами переходу. Такий автомат стає робочим СА після завантаження в нього умов переходів, що відповідають конкретному набору XML-запитів. У роботі [24] досліджено нову структуру даних – список зі зсувом (shifter list), яка застосовується саме у ПЛІС і дає змогу прискорювати процес парсингу. Обробка такої структури нагадує систолічну обробку.

Недоліки приведених підходів наступні:

заради налаштування на довільну XML-граматику апаратні схеми аналізу є надлишковими і мають великі апаратні витрати;

клас документів, що обробляються, є обмеженим.

Багатопроцесорні засоби фільтрації. Існує кілька підходів з використанням спеціалізованих паралельних архітектур для обробки XML-документів, при використанні яких прискорення досягається внаслідок виконання обробки n документів одночасно [25]. Для прискорення обробки процесори мають апаратну підтримку [18].

У роботі [26] для фільтрації запитів XML використовується графічний акселератор GPU, але його ефективність суттєво зменшується при обробці коротких запитів.

Програмно-апаратні засоби фільтрації. Є сенс застосувати ПЛІС для прискорення синтаксичного аналізу XML у програмній системі. Така ПЛІС обробляє вхідні документи і видає для кожного документа набір тегів, який далі обробляється програмно. Таким чином, до 97% вхідних запитів може бути відфільтровано з потоку [27]. На ПЛІС ефективно реалізувати декодер тегів, який прискорює роботу аналізатора SAX [22].

Отже, можна зробити такі висновки.

• Апаратні системи фільтрації XML-запитів на основі СА, реалізовані на ПЛІС, мають найбільшу швидкодію, але вони розраховані на обмежену кількість граматик і їхнє переналаштування є довготривалим.

• Апаратні системи фільтрації на основі переналаштовуваних та параметричних СА мають надмірні апаратні витрати та орієнтацію на певний клас граматик XML-документів. *n*-процесорна система здатна прискорити обробку XML-документів у *n* разів за умови організації крупнозернистого паралелізму з *n* потоками.

• В апаратно-програмних системах чітко виділяється програмна підсистема, призначена для попередньої та кінцевої обробки XML-документа, і апаратна підсистема, яка виконує розпізнавання тегів та гілок дерева документа.

• Необхідно впроваджувати більш гнучкі архітектури, спроможні забезпечити високу пропускну здатність при можливості швидкого переналаштування на довільну граматику XML.

Процесор для обробки XML-документів. Розглянувши методи і засоби прискорення фільтрації XML-документів, можна зрозуміти, що програмований процесор, який конфігурується у ПЛІС і має команди та спеціалізоване обладнання для прискорення граматичного розбору і порівняння тегів, нескладно переналаштовується на обробку XML-документів з різною структурою і має достатньо високу продуктивність. Система з таких процесорів здатна реалізувати як крупнозернисте, так і середньозернисте розпаралелювання вирішення цієї задачі. Але архітектура таких процесорів недостатньо досліджена. Для прискорення обробки XML-документів за допомогою таких конфігурованих процесорів необхідні додаткові апаратні засоби у вигляді асоціативної пам’яті для перекодування довгих ключових слів, стекової пам’яті для зберігання рівнів документа та правил граматики, компаратори тегів.

З урахуванням цих вимог розроблено архітектуру процесора для апаратно-програмної обробки XML-документів, яка належить класу стекових архітектур. Система команд стекового процесора має свої особливості, а саме операнди мають неявну адресацію, тому що зазвичай роз-

міщуються у небагатьох фіксованих регістрах стекової пам'яті і мають короткий формат. Оскільки ці команди підтримують алгоритми, що активно використовують стекову адресацію, програми, складені для такого процесора, мають мінімізовану довжину [28].

Розроблено декілька проектів стекових процесорів, які реалізовано в ПЛІС і які є доступними для відтворення [29, 30]. У роботі [30] показано, що 16-розрядний стековий процесор має програму приблизно у 2,5 рази меншої довжини, ніж програма для процесора Xilinx Microblaze [31] при реалізації протоколів обміну даними, які включають в себе парсинг пакетів. Крім того, усі стекові процесори дають змогу збільшити систему команд на кілька команд користувача.

Отже, архітектура стекового процесора дає змогу мінімізувати об'єм вбудованого матзабезпечення та одержати мінімізовані апаратні витрати внаслідок реалізації спрощених команд. Крім того, для такої архітектури нескладно розробляти компілятори, тому що зазвичай система команд є підмножиною команд мови Forth. Відомо, що ця мова є зручною для реалізації як граматичного розбору рядків, так і для інтерпретації операторів мов високого рівня. Програма мовою асемблера стекового процесора має такий самий синтаксис, як програма мовою Forth [28]. Тому доцільно розробити таку архітектуру стекового процесора, яка дає не тільки мінімізовані апаратні витрати, але й забезпечує ефективне виконання алгоритмів обробки XML-документів.

Структуру розробленого 16-розрядного процесора SM16 показано на рис. 2. Цей процесор має поширену двохстекову архітектуру [28]. Він є удосконаленою моделлю восьмирозрядного процесора SM8, який розроблено для реалізації протоколів обміну даними через послідовні інтерфейси [32]. До складу процесора входять лічильник команд PC, блок пам'яті даних Data RAM, блок пам'яті програм Program ROM, регістр команди IR, стек адрес повернення RStack, стек даних DStack, арифметико-логічний пристрій ALU. Регістри T, N – це крайні регістри стеку DStack, призначені для зберігання операндів та результату ALU. Регістр R — вершина стека RStack, який також є лічильником циклів.

Програма процесора завантажується в блок Program ROM у процесі конфігурування ПЛІС. Блок Data RAM є двохпортовим, що дає змогу завантажувати в нього XML-документи з зовнішніх пристроїв у режимі прямого доступу до пам'яті.

Регістри A, B є індексними і забезпечують ефективний доступ до рядків і масивів у Data RAM. Регістр периферійного пристрою Ri може виконувати різні функції, наприклад зв'язок з іншими процесорами системи. Блок пам'яті HTable зберігає хеш-таблицю перекодування довгих

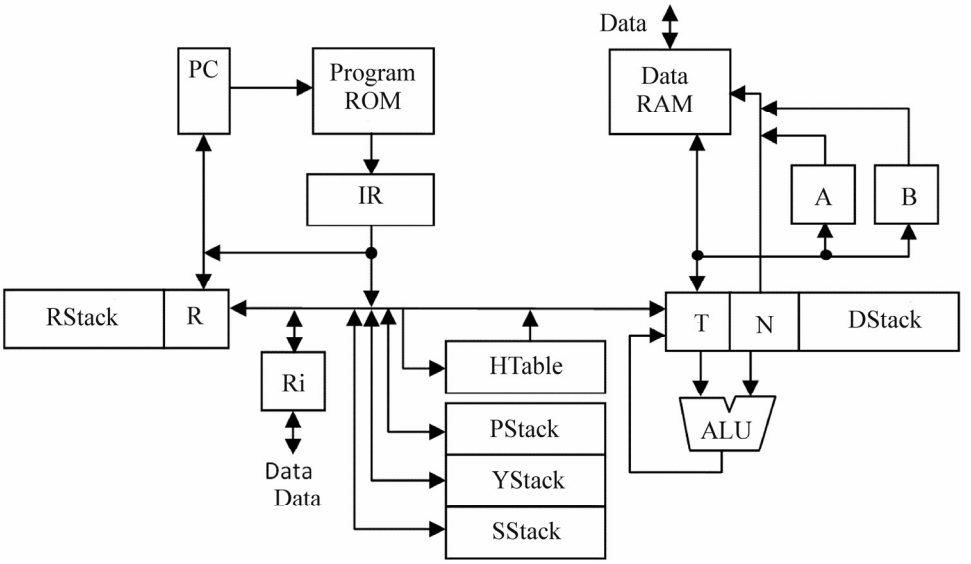

Рис. 2. Структура процесора SM16

тегів, які зустрічаються у XML-документах, у короткі номери і завантажується під час конфігурування ПЛІС. Стеки PStack, Ystack та SStack виконують таку саму функцію, як і одноіменні стеки стекового СА, що виконує алгоритм граматичного розбору. Доступ до блоків HTable, PStack, YStack та SStack виконується так само, як до регістрів периферійних пристроїв і триває лише один такт.

Команди процесора виконуються за один такт, за виключенням команд переходів, які виконуються за два такти. Ця особливість відрізняє дану архітектуру від RISC-архітектури, в якій умовний перехід зазвичай виконується більш ніж за три такти. Вона є сприятливою для реалізації алгоритмів граматичного розбору, які мають інтенсивне використання операцій переходу.

Усі команди процесора SM16 є шістнадцятирозрядними (рис. 3). Команди мають від одного до трьох полів операцій F1, F2, F3 та поле D змінної довжини, що зберігає константу, адресу переходу або зміщення цієї адреси. Архітектура стекового процесора має таку особливість: три операції, F1, F2, F3, можуть бути виконані одночасно чи послідовно в залежності від наявності безпосереднього зв’язку між ними. Наприклад, за командою :L1 HASH @ab+ L1 DJNZ над байтом у регістрі T та кодом у регістрі N виконується хеш-функція, результат якої записується у N (операція HASH), якщо [R] ≠ 0, то виконується перехід на мітку L1 та декрементується регістр R, інакше виконується вихід з циклу (операція

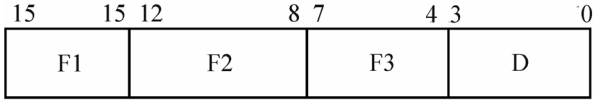

Рис. 3. Формат команди процесора SM16

DJNZ зі зміщенням D, що відповідає L1). Одночасно у регістр T з Data RAM читається черговий байт за адресою в регістрі A, після чого ця адреса інкрементується (операція @ab+).

Таким чином, використавши одну команду та витративши 2*n* тактів, рядок довжиною *n* перетворюється у хеш-код. Цей код використовується як адреса доступу до пам'яті HTable. Аналогічно за однією командою відшукується заданий символ у рядку довжиною *n* символів.

Процесор має систему переривань від подій переповнення стеків та зовнішніх подій таких, як кінець запису у Data RAM чи периферійний регістр Ri. Під час векторизації переривання у регістр IR записується штучно створена команда виклику підпрограми з відповідною адресою-вектором переривання. Оскільки у стекового процесора контекст має мінімальний об'єм, накладні витрати на переривання також є незначними.

Програмування стекового процесора зазвичай виконується стилем шитого коду (threaded code), коли програма виглядає як послідовність викликів підпрограм. Це дає змогу одержати програми мінімізованої довжини, що є важливим при реалізації процесора у ПЛІС. Можливість вставлення операції повернення з підпрограми, суміщаючи її з операцією умовного переходу, у більшості команд процесора SM16 дає змогу значно скоротити довжину підпрограм та їх тривалість. Через велику кількість команд виклику підпрограми, умовного переходу та читання пам'яті у програмах обробки XML-документів середня тривалість виконання однієї команди в даній архітектурі складає півтора такти.

У табл. 2 наведено результати синтезу процесора SM16 у ПЛІС Xilinx Spartan-6 при встановленні параметрів оптимізації за апаратурними витратами. Для порівняння дано також параметри перелічених вище процесорів, синтезованих у таких самих умовах. Додано для порівняння мікропроцесор з поширеною архітектурою MSP430 такої самої розрядності. Апаратні витрати SM16 вказано для мінімальної конфігурації ядра та ядра, до якого додано три стеки та хеш-таблиця.

Аналізуючи табл. 2, бачимо, що процесор SM16 при значно більших апаратних витратах має таку саму продуктивність, як процесор J1, значно більшу швидкодію, ніж процесор MSP430, і дещо програє процесору Microblaze. Останнє пояснюється тим, що архітектуру швидкого проце-

сора адаптовано до структури ПЛІС самою фірмою-виробником цих ПЛІС і подано у формі, яка сприяє одержанню мінімального критичного шляху. Але слід зазначити, що SM16 має більшу систему команд ніж у J1. При цьому три операції виконуються паралельно і адаптовані до обробки XML-документів. Окрим того, SM16 має систему переривань.

Розроблений асемблер компілює текст програми у машинні коди, які заносяться у пам'ять Program ROM і далі потрапляють у поле змісту відповідного блоку пам'яті файлу прошивки ПЛІС. Для прискорення моделювання розроблених програм використовується програмний симулятор процесора.

Система для апаратно-програмної обробки XML-документів. Процесор SM16 спроможний виконувати граматичний розбір XML-документів за алгоритмами, описаними вище. Для налаштування на іншу граматику слід перекомпілювати опис граматики у програму, яка вбудовується в сегмент відповідного блока пам'яті файлу конфігурації ПЛІС. Такий процес триває приблизно одну хвилину і є значно коротшим за процес трансляції опису граматики у СА, який конфігурується у ПЛІС.

Недостатня швидкодія одного процесора компенсується використанням багатопроцесорної паралельної системи. Така система складається з десятків процесорних елементів (ПЕ) SM16, які виконують функції виконавчих процесорів та ведучого процесора, що розподіляє XML-документи серед цих ПЕ для обробки. Наприклад, в ПЛІС Xilinx Zynq-7 ведучим процесором стає вбудоване процесорне ядро з архітектурою Cortex-A9 і в цю ПЛІС може бути розміщено від 20 до 300 ПЕ SM16 в залежності від кількості конфігурованих логічних блоків, яку вона вміщує.

Початковими даними для настроювання системи є набір граматик XML-запитів, наприклад описаних мовою XPath [10]. Алгоритми граматичного розбору цих запитів, описані вище, автоматично конвертуються

Таблиця 2. **Параметри процесорів при їх конфігуруванні у ПЛІС**

Мікроконтролер	Розрядність	Апаратні витрати, LUT	Максимальна тактова частота, МГц	Продуктивність
b16-small [29]	16	280	100	50 MIPS
J1 [30]	16	342	106	70 MIPS
MSP430 [33]	16	1 240	65	25 MIPS
Microblaze [31]	32	2 046	130	174 DMIPS
SM8 [34]	8	181	140	94 MIPS
SM16	16	477—580	105	70 MIPS

у програми для ПЕ. При цьому складається загальна бібліотека підпрограм елементарних операцій, що відповідають обробці тегів різного типу, а програма обробки запиту конкретного типу формується стилем шитого коду з викликами відповідних підпрограм. Для скорочення об'єму програми застосовується такий самий підхід до формування автомата пошуку співпадінь, як в роботі [17].

Одержані програми завантажуються у ПЕ під час конфігурування ПЛІС або з ведучого процесора у режимі прямого доступу до пам'яті Program ROM. Таким чином, переналаштування системи на прийом інших XML-запитів відбувається значно швидше, ніж у повністю апаратній системі.

Для оцінки ефективності системи варто порівняти її з системою обробки XML-запитів, запропонованій у роботі [35]. До складу цієї системи входять ведучий процесор з архітектурою Cortex-A9 і апаратний прискорювач, зконфігурований у ПЛІС. В результаті випробування встановлено, що такий апаратно-програмний парсер при парсингу XML-запитів у форматі EXI зменшує тривалість парсингу на 23%, а чисто апаратний парсер — на 95%. При цьому конфігурована апаратура парсера працює на частоті 50 МГц і має апаратні витрати 12,8 тис. LUT. При таких самих апаратних витратах запропонована система включатиме приблизно 20 ПЕ і забезпечуватиме не гіршу або кращу пропускну здатність, маючи такі переваги, як швидка переналаштовуваність та можливість вирішувати задачі широкого класу.

Висновки

За результатами дослідження встановлено, що при реалізації парсерів, керованих подіями, найбільшу швидкодію мають апаратні пристрої, в яких реалізовано скінченні автомати, налаштовані на пошук співпадіння XML-запитів з вимогами граматики цих запитів. Зазвичай такі пристрої реалізовано у ПЛІС, але вони вимагають тривалого переналаштування при зміні граматики. Апаратно реалізовані скінченні автомати, які є універсальними стосовно певної множини граматик, мають надмірні апаратні витрати. Ефективність реалізації алгоритмів аналізу XML-документів у багатопроцесорній системі обмежена тим, що операції алгоритмів граматичного розбору виконуються повільно з використанням існуючих систем команд, тобто є невідповідність між архітектурою ПЕ та операціями алгоритмів.

Прийнято рішення про розробку процесорного ядра SM16, архітектура якого відповідає алгоритмам обробки XML-документів. Базовою архі-

тектурою ядра обрано стекову архітектуру, яка забезпечує малі апаратні витрати, мінімізовану довжину програмних кодів при реалізації в ПЛІС, а також ефективну реалізацію стекових алгоритмів, до яких належать більшість алгоритмів обробки документів з деревовидною структурою. Для збільшення ефективності виконання алгоритмів до архітектури ядра додано три блоки стекової пам'яті, хеш-таблиця та команди, що прискорюють виконання певних операцій. Обраний формат команд дозволяє виконувати кілька операцій паралельно і забезпечує мінімізовану довжину програм.

Запропонована апаратно-програмна система на основі ПЛІС має ведучий процесор з архітектурою Cortex-A9 та до сотні виконавчих ПЕ SM16. Така система здатна не тільки ефективно обробляти XML-документи, але й швидко переналаштовуватись на обробку документів з іншими граматиками.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. *Head M. R., Govindaraju M., van Engelen R., Zhang W.* Benchmarking XML processors for applications in grid Web services // Proc. of the 2006 ACM/IEEE conference on Supercomputing. [Electronic resource] Electronic data. Mode of access: https://www.doi.org/10.1109/SC.2006.14 (viewed on January 1, 2020).
2. *Driscoll D., Mensch A.* Devices profile for web services version 1.1 [Electronic resource] / OASIS. Electronic data. Mode of access: http://docs.oasis-open.org/ws-dd/dpws/wsdd-dpws-1.1-spec.html (viewed on November 9, 2019). Title from screen.
3. *Schneider J., Kamiya T.* Efficient XML Interchange (EXI) Format 1.0 [Electronic resource] / World Wide Web Consortium (W3C). Electronic data. Mode of access: http://www.w3.org/TR/exi/ (viewed on November 9, 2019). Title form screen.
4. *Apparao P., Bhat M.* A detailed look at the characteristics of XML parsing. // BEACON'04: 1st Workshop on Building Block Engine Architectures for Computers and Networks, 2004.
5. *Mattias N., Jasmi J.* XML Parsing: A Threat to Database Performance. // Proc. of the 20th International Conference on Information and Knowledge Management CIKM '03, 2003, p. 175–178.
6. SAX Parsing Model [Electronic resource] : [Web-site]. Electronic data. Mode of access: http://sax.sourceforge.net (viewed on November 11, 2019). Title from screen.
7. Document object model (DOM) level 2 core specification [Electronic resource] / W3C. Electronic data. Mode of access: http://www.w3.org/TR/DOM-Level-2-Core (viewed on November 11, 2019). Title from screen.
8. *Comon H., Dauchet M., Gilleron R., Jacquemard F.et al.* Tree Automata Techniques and Applications [Electronic resource]: [Web-site]. Electronic data. Mode of access: http://tata.gforge.inria.fr/ (viewed on November 11, 2019). Title from screen.
9. *Murata M., Lee D., Mani M., Kawaguchi K.* Taxonomy of XML schema languages using formal language theory. // ACM Transactions on Internet Technology, 2005, Vol. 5, Issue 4, p. 660–704.
10. XML Path Language Version 1.0 [Electronic resource]: [Web-site]. Electronic data. Mode of access: http://www.w3.org/TR/xpath (viewed on November 11, 2019). Title from screen.

11. *Chiu K., Devadithya T., Lu W., Slominski A*. A binary XML for scientific applications. // Proc. First International Conference on e-Science and Grid Computing (e-Science'05). IEEE, 2005, p. 336-343.
12. *Lu W., Chiu K., Pan Y.* A Parallel Approach to XML Parsing. // Proc. of the 7th IEEE/ ACM International Conference on Grid Computing, 2006, p. 223-230.
13. *Head M. R., Govindaraju M.* Approaching a Parallelized XML Parser Optimized for Multi-Core Processor. // Proc. of the 2007 Workshop on Service-Oriented Computing Performance: Aspects Issues and Approaches (SOCP'07), 2007, p. 17-22.
14. *Altinel M., Franklin M. J.* Efficient Filtering of XML Documents for Selective Dissemination of Information. // Proc. of the 26th International Conference on Very Large Data Bases (VLDB'00), 2000, p. 53-64.
15. *Diao Y., Altinel M., Franklin M. J. et al.* Path sharing and predicate evaluation for high-performance XML filtering. // ACM Trans. on Database Systems (TODS), 2003, Vol. 28, p. 467-516.
16. *Green T. J., Gupta A., Miklau G. et al.* Processing XML streams with deterministic automata and stream indexes. // ACM Trans.on Database Systems (TODS), 2004, p. 752-788.
17. *Silvasti P.* XML-Document-Filtering Automaton. // Proc. of the Very Large Data Base Endowment (VLDB Endowment '08), 2008, Vol. 1, Issue 2, p. 1666-1671.
18. *Lunteren J.V., Engbersen T., Bostian J. et al.* XML accelerator engine // 1st International Workshop on High Performance XML Processing, 2004. [Electronic resource] Electronic data. Mode of access: http://wam.inrialpes.fr/www-workshop2004/ZuXA_final_paper.pdf (viewed on January 1, 2020).
19. *El-Hassan F., Ionescu D.* SCBXP: An efficient hardware-based XML parsing technique. // 5th Southern Conference on Programmable Logic (SPL), 2009, p. 45-50.
20. *Mueller R., Teubner J., Alonso G.* Streams on wires — a query compiler for FPGAs. // Proc. of the Very Large Data Base Endowment (VLDB Endowment '09), 2009, Vol. 1, Issue 2, p. 229-240.
21. *Moussalli R., Salloum M., Najjar W., Tsotras V*. Massively Parallel XML Twig Filtering Using Dynamic Programming on FPGAs. // Proc. of the IEEE 27th International Conference on Data Engineering (ICDE'11), 2011, p. 948-959.
22. *Mitra A., Vieira M., Bakalov P.et al.* Boosting XML Filtering with a Scalable FPGA-based Architecture. // Proc. of the CIDR 2009 - 4th Biennal Conference on Innovative Data Systems Research, 2009. [Electronic resource] Electronic data. Mode of access: https:// arxiv.org/abs/0909.1781 (viewed on January 1, 2020).
23. *Teubner J., Woods L., Nie C.* XLynx — an FPGA-based XML filter for hybrid XQuery processing. // ACM Transactions on Database Systems (TODS), 2013, Vol. 38, Issue 4.
24. *Woods L., Alonso G., Teubner J.* Parallelizing data processing on FPGAs with shifter lists. // ACM Transactions on Reconfigurable Technology and Systems (TRETS). Special Section on FPL 2013, 2015, Vol. 8, Issue 2.
25. *Letz S., Zedler M., Thierer T. et al.* XML offload and acceleration with Cell broadband engine. // XTech: Building Web 2.0, 2006. [Electronic resource] Electronic data. Mode of access: https://doi.org/10.1007/978-3-642-11515-8_12 (viewed on January 1, 2020).
26. *Moussalli R., Halstead R.et al.* Efficient XML Path Filtering Using GPUs. // Proc. of the 2nd International Workshop on Accelerating Data Management Systems (ADMS 2011),

2011. [Electronic resource] Electronic data. Mode of access: https://www. Researchgate.net/publication/257631377_Efficient_XML_Path_Filtering_Using_GPUs (viewed on January 1, 2020).
27. *Fischer P., Teubner J*. MXQuery with hardware acceleration. // Proc. of the IEEE 28th International Conference on Data Engineering (ICDE), 2012, p. 1293-1296.
28. *Koopman P.* Stack computers: the new wave. CA: Ellis Horwood, Mountain View Press. 1989, 234 p.
29. *Paysan B.* b16-small — Less is More // Proc. of the EuroForth 2004, 2006, p. 1-8.
30. *Bowman J., Garage W.* J1: a small Forth CPU Core for FPGAs // Proc. of the EuroForth'2010, 2010, p. 1-4.
31. *Kale V*. Using the MicroBlaze Processor to Accelerate Cost-Sensitive Embedded System Development [Electronic resource] Xilinx, WP469 (v1.0.1). Electronic data. Mode of access: https://www.xilinx.com/products/design-tools/microblaze.html#documentation (viewed on November 11, 2019). Title from screen.
32. *Sergiyenko A., Molchanov O., Orlova M.* Nano-Processor for the Small Tasks. // IEEE 39th International Conference on Electronics and Nanotechnology (ELNANO), 2019, p. 674-677.
33. *Girard O.* OpenMSP430 [Electronic resource] : [Web-site]. Electronic data. Mode of access: http://opencores.org (viewed on November 11, 2019). Title from screen.
34. *Molchanov O., Orlova M., Sergiyenko A.* Software/Hardware Co-design of the Microprocessor for the Serial Port Communications. // In: Hu Z., Petoukhov S., Dychka I., He M. (eds) Advances in Computer Science for Engineering and Education II. ICCSEEA 2019. Advances in Intelligent Systems and Computing, Springer, 2020, p. 238-246.
35. *Altmann V., Skodzik J., Danielis P.et al.* Real-Time Capable Hardware-based Parser for Efficient XML Interchange. // Proc. of the 9th International Symposium on Communication Systems, Networks & Digital Sign (CSNDSP), 2014, p. 415-420.

Отримано 25.11.19

REFERENCES

1. Head, M.R., Govindaraju, M., van Engelen, R. and Zhang, W. (2006), "Benchmarking XML processors for applications in grid Web services", *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, available at: https://www.doi.org/ 10.1109/ SC.2006.14 (accessed: 1 January 2020).
2. Driscoll, D. and Mensch, A. (2009), "Devices profile for web services version 1.1", available at: http://docs.oasis-open.org/ws-dd/dpws/wsdd-dpws-1.1-spec.html (accessed 9 November 2019).
3. Schneider, J. and Kamiya, T. (2011), "Efficient XML Interchange (EXI) Format 1.0", available at: http://www.w3.org/TR/exi/ (accessed 9 November 2019).
4. Apparao, P. and Bhat, M. (2004), "A detailed look at the characteristics of XML parsing", *BEACON-1: 1st Workshop on Building Block Engine Architectures for Computers and Networks*.
5. Mattias, N. and Jasmi, J. (2003), "XML Parsing: A Threat to Database Performance", *Proceedings of the 20th International Conference on Information and Knowledge Management CIKM '03*, pp. 175-178.
6. SorceForge. (2002), "SAX Parsing Model", available at: http://sax.sourceforge.net (accessed 11 November 2019).
7. W3C. (2000), "Document object model (DOM) level 2 core specification", available at: http://www.w3.org/TR/DOM-Level-2-Core (accessed 11 November 2019).

8. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Lueding, C., Tison, S. and Tommasi, M. (2008), "Tree Automata Techniques and Applications", available at: http://tata.gforge.inria.fr/ (accessed 11 November 2019).
9. Murata, M., Lee, D., Mani, M. and Kawaguchi, K. (2005), "Taxonomy of XML schema languages using formal language theory", *ACM Transactions on Internet Technology*, Vol. 5, Iss. 4, pp. 660-704.
10. W3C. (1999), "XML Path Language Version 1.0", available at: http://www.w3.org/TR/xpath (accessed 11 November 2019).
11. Chiu K., Devadithya T., Lu W. and Slominski A. (2005), "A binary XML for scientific applications", *Proceedings of the IEEE 1st International Conference on e-Science and Grid Computing (e-Science'05)*, pp. 336-343.
12. Lu, W., Chiu, K. and Pan, Y. (2006), "A Parallel Approach to XML Parsing", *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, pp. 223-230.
13. Head, M.R. and Govindaraju, M. (2007), "Approaching a Parallelized XML Parser Optimized for Multi-Core Processor", *Proceedings of the 2007 Workshop on Service-Oriented Computing Performance: Aspects Issues and Approaches (SOCP'07)*, pp. 17-22.
14. Altinel, M. and Franklin, M.J. (2000), "Efficient Filtering of XML Documents for Selective Dissemination of Information", *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB'00)*, pp. 53-64.
15. Diao, Y., Altinel, M., Franklin, M.J., Zhang, H. and Fischer, P. (2003), "Path sharing and predicate evaluation for high-performance XML filtering", *ACM Trans. on Database Systems (TODS)*, Vol. 28, pp. 467-516.
16. Green, T. J., Gupta, A., Miklau, G., Onizuka, M. and Suciu, D. (2004), "Processing XML streams with deterministic automata and stream indexes", *ACM Trans.on Database Systems (TODS)*, pp. 752-788.
17. Silvasti, P. (2008), "XML-Document-Filtering Automaton", *Proceedings of the Very Large Data Base Endowment (VLDB Endowment '08)*, Vol. 1, Iss. 2, pp. 1666-1671.
18. Lunteren, J.V., Engbersen, T., Bostian, J., Carey, B. and Larsson, C. (2004), "XML accelerator engine", *Proceeding of the 1st International Workshop on High Performance XML*, available at: http://wam.irialpes.fr/www-workshop2004/ZuXA_final_paper.pdf (accessed 1 January 2020).
19. El-Hassan, F. and Ionescu, D. (2009), "SCBXP: An efficient hardware-based XML parsing technique", *Proceeding of the 5th Southern Conference on Programmable Logic (SPL)*, pp. 45-50.
20. Mueller, R., Teubner, J. and Alonso, G. (2009), "Streams on wires — a query compiler for FPGAs", *Proceedings of the Very Large Data Base Endowment (VLDB Endowment '09)*, Vol. 1, Iss. 2, pp. 229-240.
21. Moussalli, R., Salloum, M., Najjar, W. and Tsotras, V. (2011), "Massively Parallel XML Twig Filtering Using Dynamic Programming on FPGAs", *Proceedings of the IEEE 27th International Conference on Data Engineering (ICDE'11)*, pp. 948-959.
22. Mitra, A., Vieira, M., Bakalov, P., Najjar, W. and Tsotras, V. (2009), "Boosting XML Filtering with a Scalable FPGA-based Architecture", *Proceedings of the CIDR 2009 - 4th Biennal Conference on Innovative Data Systems Research*, available at: https://arxiv.org/abs/0909.1781 (accessed: 1 January 2020).
23. Teubner, J., Woods, L. and Nie, C. (2013), "XLynx — an FPGA-based XML filter for hybrid XQuery processing", *ACM Transactions on Database Systems (TODS)*, Vol. 38, Iss. 4.

24. Woods, L., Alonso, G. and Teubner, J. (2015), "Parallelizing data processing on FPGAs with shifter lists", *ACM Transactions on Reconfigurable Technology and Systems (TRETS) - Special Section on FPL 2013*, Vol. 8, Iss. 2.
25. Letz, S., Zedler, M., Thierer, T., Schutz, M., Roth, J. and Seiffert, R. (2006), "XML offload and acceleration with Cell broadband engine", *XTech: Building Web 2.0*, available at: https://doi.org/10.1007/978-3-642-11515-8_12 (accessed: 1 January 2020).
26. Moussalli, R., Halstead, R., Solloum, M., Najjar, W. and Tsotras, V. (2011), "Efficient XML Path Filtering Using GPUs", *Proceedings of the 2nd International Workshop on Accelerating Data Management Systems (ADMS 2011)*, available at: https://www. researchgate.net/publication/257631377_Efficient_XML_Path_Filtering_Using_GPUs (accessed: 1 January 2020).
27. Fischer, P. and Teubner, J. (2012), "MXQuery with hardware acceleration", *Proceedings of the IEEE 28th International Conference on Data Engineering (ICDE)*, pp. 1293-1296.
28. Koopman, P. (1989), Stack computers: the new wave, Ellis Horwood, Mountain View Press.
29. Paysan, B. (2006), "b16-small — Less is More", *Proceedings of the EuroForth 2004*, pp. 1-8.
30. Bowman, J. and Garage, W. (2010), "J1: a small Forth CPU Core for FPGAs", *Proceedings of the EuroForth'2010*, pp. 1-4.
31. Kale, V. (2016), "Using the MicroBlaze Processor to Accelerate Cost-Sensitive Embedded System Development", *Xilinx, WP469 (v1.0.1)*, available at: https://www.xilinx.com/ products/design-tools/microblaze.html#documentation (accessed 11 November 2019).
32. Sergiyenko, A., Molchanov, O. and Orlova, M. (2019), "Nano-Processor for the Small Tasks", *Proceeding of the 39th 2019 IEEE International Conference on Electronics and Nanotechnology (ELNANO)*, pp. 674-677.
33. Girard, O. (2013), "OpenMSP430", *OpenCores, Rev. 1.13*, available at: http:// opencores.org (accessed 11 November 2019).
34. Molchanov, O., Orlova, M. and Sergiyenko, A. (2020), "Software/Hardware Co-design of the Microprocessor for the Serial Port Communications", *Springer*, pp. 238-246.
35. Altmann, V., Skodzik, J., Danielis, P., Van, N.P., Golatowski, F. and Timmermann, D. (2014), "Real-Time Capable Hardware-based Parser for Efficient XML Interchange", *Proceedings of the 9th International Symposium on Communication Systems, Networks & Digital Sign (CSNDSP)*, pp. 415-420.

Received 25.11.19

А.М. Сергиенко, М.М. Орлова, О.А. Молчанов

АППАРАТНО-ПРОГРАММНАЯ ОБРАБОТКА XML-ДОКУМЕНТОВ

Рассмотрены существующие алгоритмы и средства обработки XML-документов. Обоснована необходимость использования высокопроизводительных устройств анализа XML-запросов, способных быстро перенастраиваться на другие грамматики. Разработано процессорное ядро SM16, которое эффективно выполняет стековые алгоритмы парсинга XML-документов и реализуется в программируемых логических интегральных схемах (ПЛИС). Процессорное ядро имеет архитектуру стекового процессора, к которому добавлено три блока стековой памяти, хеш-таблица и команды, ускоряющие выполнение операций

парсинга. Предложена аппаратно-программная система на основе ПЛИС, имеющая ведущий процессор и от десятков до сотен исполнительных процессорных элементов SM16. Эта система не только эффективно обрабатывает XML-документы, но и быстро перенастраивается на обработку документов с другими грамматиками.

К л ю ч е в ы е с л о в а: XML, парсер, стековый процессор, грамматика, стековый автомат.

A.M. Sergiyenko, M.M. Orlova, O.A. Molchanov

HARDWARE-SOFTWARE XML-DOCUMENTS PROCESSING

Existing algorithms and tools for XML-documents processing are reviewed in this article. A need in highly productive devices that analyze XML-requests and that can be easily reconfigured for different grammars is determined. SM16 processor core is developed. Its architecture effectively evaluates stack-based parsing algorithms and is implemented on field programmable gate arrays (FPGA). Processor architecture is based on stack processor architecture with three additional stack memory blocks, hash-table and instructions that accelerate execution of parsing operations. We propose hardware-software FPGA-based system, which has main processor and tens to hundreds of SM16 executive processor elements. This system efficiently processes XML-documents and can be easily reconfiguration to process documents with different grammars.

K e y w o r d s: XML, parser, stack processor, grammar, stack automaton.

СЕРГІЄНКО Анатолій Михайлович, д-р техн. наук, професор, професор кафедри обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського». У 1981 р. закінчив Київський політехнічний інститут. Область наукових досліджень — цифрова обробка сигналів, архітектура комп'ютерів, високорівневий синтез.

ОРЛОВА Марія Миколаївна, канд. техн. наук, доцент, доцент кафедри системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського». У 1978 р. закінчила Київський політехнічний інститут. Область наукових досліджень — комп'ютерні мережі, сучасні напрямки розвитку комп'ютерних мереж, мережні інформаційні технології, безпроводові комп'ютерні мережі 5G та 6G.

МОЛЧАНОВ Олексій Андрійович, аспірант кафедри системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського», котрий закінчив у 2016 р. Область наукових досліджень — архітектура комп'ютерів, компілятори.