УДК 004.4'24 https://doi.org/10.15407/pp2020.01.017

А.Ю. Дорошенко, О.А. Яценко

АВТОМАТИЗОВАНЕ ПРОЕКТУВАННЯ ПРОГРАМ ДЛЯ ПЛАТФОРМИ .NET, ЩО ВИКОРИСТОВУЮТЬ БІБЛІОТЕКУ ПАРАЛЕЛЬНИХ ЗАДАЧ

Виконане налаштування алгебро-алгоритмічного інструментарію на формалізоване проектування та синтез паралельних програм мовою C# для платформи .NET, що використовують засоби бібліотеки паралельних задач TPL. Згадана бібліотека підвищує продуктивність праці розробників за рахунок спрощення процедури додавання паралелізму в програму та динамічно масштабує ступінь паралелізму для найбільш ефективного використання усіх доступних процесорів. В основу пропонованого підходу покладені мова САА-схем, перевагою якої є простота в навчанні й використанні, а також метод конструювання синтаксично правильних програм, що виключає можливість появи синтаксичних помилок у процесі проектування схем. Проведено експеримент з виконання згенерованих за допомогою розробленого інструментарію прикладів паралельних програм на багатоядерному процесорі.

Ключові слова: автоматизоване проектування програм, алгебра алгоритмів, багатопотоковість, бібліотека паралельних задач, паралельні обчислення, синтез програм, TPL.

Вступ

Необхідність у підвищенні продуктивності програмного забезпечення для вирішення трудомістких задач, з одного боку, і нові можливості розпаралелювання обчислень, що надаються багатоядерною архітектурою сучасних мікропроцесорів – з іншого, спонукає до створення спеціалізованих інструментальних засобів для розробки паралельних програм для таких архітектур. Головним способом підвищення продуктивності програм для згаданих платформ є розпаралелювання програм із використанням багатопотоковості. У попередніх роботах [1–5] запропоновані теорія, методологія та інструментарій для автоматизованого проектування, синтезу та перетворення послідовних та паралельних програм, що ґрунтуються на засобах алгебр алгоритмів та переписувальних правил. Зокрема, розглянуті засоби автоматизованої розробки програм для платформи .NET [3–5]; для розпаралелювання програм використовувалися стандартні засоби роботи з потоками.

Одним із способів подальшого підвищення ефективності багатопотокових програм є використання бібліотеки паралельних задач TPL (Task Parallel Library) [6, 7]. Мета TPL полягає у підвищенні продуктивності праці розробників за рахунок спрощення процедури додавання паралелізму в програмні застосунки. TPL динамічно масштабує ступінь паралелізму для найбільш ефективного використання усіх доступних процесорів. Крім того, у бібліотеці паралельних задач здійснюється секціонування роботи, планування потоків у пулі, підтримка відміни, керування станом і виконуються інші низькорівневі задачі. Використовуючи бібліотеку паралельних задач, можна підвищити продуктивність коду, зосередившись на роботі, для якої призначена програма.

У даній роботі запропонований подальший розвиток алгебро-алгоритмічного інструментарію на автоматизоване проектування та генерацію паралельних програм мовою C#, що використовують TPL. Проведено експеримент з виконання згенерованих за допомогою розробленого інструментарію прикладів програм на багатоядерному процесорі.

1. Алгебро-алгоритмічні засоби проектування паралельних програм

Пропонований підхід до конструювання програм ґрунтується на апараті систем алгоритмічних алгебр (САА) та їх модифікацій [1, 2] (САА-М), призначених для формалізації процесів мультиобробки, що виникають при проектуванні програмного забезпечення в мультипроцесорних

системах. САА-М покладено в основу розроблених інструментальних засобів автоматизованого проектування та генерації програм [3–5].

1.1. Системи алгоритмічних алгебр. Модифіковані САА – двоосновна алгебра $GA =< Pr, Op; \Omega_{GA} >$, де Pr – множина логічних умов (предикатів); Op – множина операторів; Ω_{GA} – сигнатура, що складається з логічних операцій (диз'юнкції, кон'юнкції, заперечення) та операторних операцій (композиції, альтернативи, циклу та ін.), що будуть розглянуті далі.

На САА-М грунтується алгоритмічна мова САА/1 [1, 2], призначена для багаторівневого структурного проектування й документування послідовних та паралельних алгоритмів і програм. Перевагою її використання є подання алгоритмів у природно-лінгвістичній формі. Специфікації операторів мовою САА/1 називаються САА-схемами.

Предикати та оператори САА-М поділяються на базисні та складені. Базисні елементи вважаються атомарними, неподільними абстракціями та пов'язані з предметною областю алгоритму, що конструюється.

Складені умови будуються з базисних на основі таких узагальнених булевих операцій [1]:

1) диз'юнкція:

"*condition* 1" OR "*condition* 2";

2) кон'юнкція:

"*condition* 1" AND "*condition* 2";

3) заперечення:

NOT "*condition* 1".

Складені оператори будуються на основі базисних предикатів і операторів та операцій послідовного й паралельного виконання:

1) послідовне виконання операторів (композиція):

"*operator* 1";
"*operator* 2";

2) умовний оператор (альтернатива):

IF '*condition*' THEN
  "*operator* 1"
ELSE
  "*operator* 2"
END IF;

3) цикл типу while:

WHILE '*condition*'
LOOP "*operator*"
END OF LOOP;

4) цикл типу for:

FOR (*counter* FROM *start* TO *fin*)
LOOP "*operator*"
END OF LOOP;

5) паралельне виконання *n* операторів (асинхронна диз'юнкція):

PARALLEL(*i* = 0, ..., *n*)
(
  "*operator i*"
);

6) контрольна точка, яка встановлює значення "істина" для умови '*condition*':

CP '*condition*';

7) синхронізатор, що виконує затримку обчислень доти, поки значення вказаної умови не стане істинним:

WAIT '*condition*',

де '*condition*' може бути пов'язана з умовами, вказаними у контрольних точках.

Приклад 1. Далі як ілюстрація наведено фрагмент послідовної САА-схеми знаходження кількості простих чисел на відрізку [*start, end*]. Схема містить складений оператор "SequentialPrimes (*start, end*)" та умову 'Is prime (*number*)', що відповідають підпрограмам (методам) у цільовій

мові програмування. У схемі виконується послідовна перевірка кожного непарного числа, чи ділиться воно на менші непарні числа. Кількість знайдених чисел зберігається у змінній *result*.

```
"SequentialPrimes(start, end)"
==== "Declare a variable (result)
       of type (long) = (0)";
     FOR (number FROM start TO end – 1)
       IF 'Is prime (number)'
       THEN "Increase (result) by (1)"
       END IF
     END OF LOOP;
     "Return value (result)";

'Is prime (number)'
==== IF (number = 2)
     THEN "Return value (true)"
     END IF;
     IF (number % 2 = 0)
     THEN "Return value (false)"
     END IF;
     "Declare a variable (divisor)
       of type (long) = (3)";
     WHILE NOT (divisor >= number / 2)
     LOOP
       IF (number % divisor = 0)
       THEN "Return value (false)"
       END IF;
       "Increase (divisor) by (2)"
     END OF LOOP;
     "Return value (true)";
```

Приклад 2. Розглянемо далі паралельний варіант алгоритму знаходження кількості простих чисел:

```
"StandardThreadsPrimes(start, end)"
==== "Declare a variable (result)
       of type (long) = (0)";
     "Declare a variable (range)
       of type (long) = (end – start)";
     "Declare a variable (numberOfThreads)
       of type (long)";
      (numberOfThreads :=
       "Get the number of processors");
     "Declare a variable (chunkSize) of type
       (long) = (range / numberOfThreads)";
     PARALLEL(i = 0, ...,
                  numberOfThreads – 1)
      ( "Declare the list of variables
        (chunkStart, chunkEnd)
        of type (long)";
       (chunkStart := start + i * chunkSize);
       IF (i = numberOfThreads – 1)
       THEN (chunkEnd := end)
       ELSE
         (chunkEnd := chunkStart +
          chunkSize)
       END IF;
       FOR (number FROM chunkStart TO
            chunkEnd – 1)
         IF 'Is prime (number)'
         THEN "Increase (result) by (1)
                interlocked"
         END IF
       END OF LOOP;
       CP 'Thread (i) completed work'
      );
     WAIT 'All threads completed work';
     "Return value (result)";
```

Обчислення у даному алгоритмі здійснюються паралельними гілками, кожна з яких обробляє свою порцію даних на ділянці [*chunkStart*, *chunkEnd*], на які порівну розділений відрізок [*start*, *end*]. Кількість потоків дорівнює кількості доступних у системі процесорів. Синхронізація гілок виконується за допомогою контрольних точок та синхронізатора. Сумісно використовуваним потоками ресурсом є змінна *result*. На відміну від послідовного алгоритму, у паралельному для збільшення значення цієї змінної використовується оператор інкременту з блокуванням, а саме: "Increase (*result*) by (1) interlocked".

1.2. Інструментарій автоматизованої розробки програм. Розроблений інтегрований інструментарій проектування та синтезу програм (ІПС) ґрунтується на використанні розглянутих засобів САА-М та методу діалогового конструювання синтаксично правильних програм (ДСП-методу) [1, 2]. На відміну від традиційних синтаксичних аналізаторів, ДСП-метод орієнтований не на пошук і виправлення синтаксичних помилок, а на виключення можливості їх появи в процесі побудови алгоритму. Метод полягає у порівневому конструюванні схем зверху вниз за допо-

могою суперпозиції мовних конструкцій САА-М. На кожному кроці конструювання система надає користувачу на вибір лише ті конструкції, підстановка яких у текст алгоритму, що проектується, не порушує синтаксичну правильність схеми. На основі побудованої схеми алгоритму виконується автоматична генерація тексту програми цільовою мовою програмування. Відображення операцій САА-М у текст мовою програмування подане у вигляді шаблонів і зберігається в базі даних інструментарію. На рис. 1 показано копію екрану системи ІПС із сконструйованим паралельним алгоритмом StandardThreadsPrimes.

Для автоматизації виконання трансформацій алгоритмів система ІПС може застосовуватися спільно з системою TermWare [3–5], що ґрунтується на техніці переписувальних правил.

2. Налаштування алгебро-алгоритмічних засобів на генерацію паралельних програм, що використовують бібліотеку TPL

Бібліотека паралельних задач (TPL) є набором відкритих типів і API-інтерфейсів в просторах імен System.Threading і System.Threading.Tasks в рамках платформи .NET [6]. Ця бібліотека дозволяє автоматично розподіляти навантаження застосувань між доступними процесорами в динамічному режимі, використовуючи пул потоків CLR та конструкції Parallel.For і Parallel.ForEach [7]. Бібліотека TPL виконує розподіл роботи, планування потоків, управління станом та інші низькорівневі задачі. В результаті з'являється можливість максимізувати продуктивність застосувань .NET, не маючи справи із складнощами безпосередньої роботи з потоками.

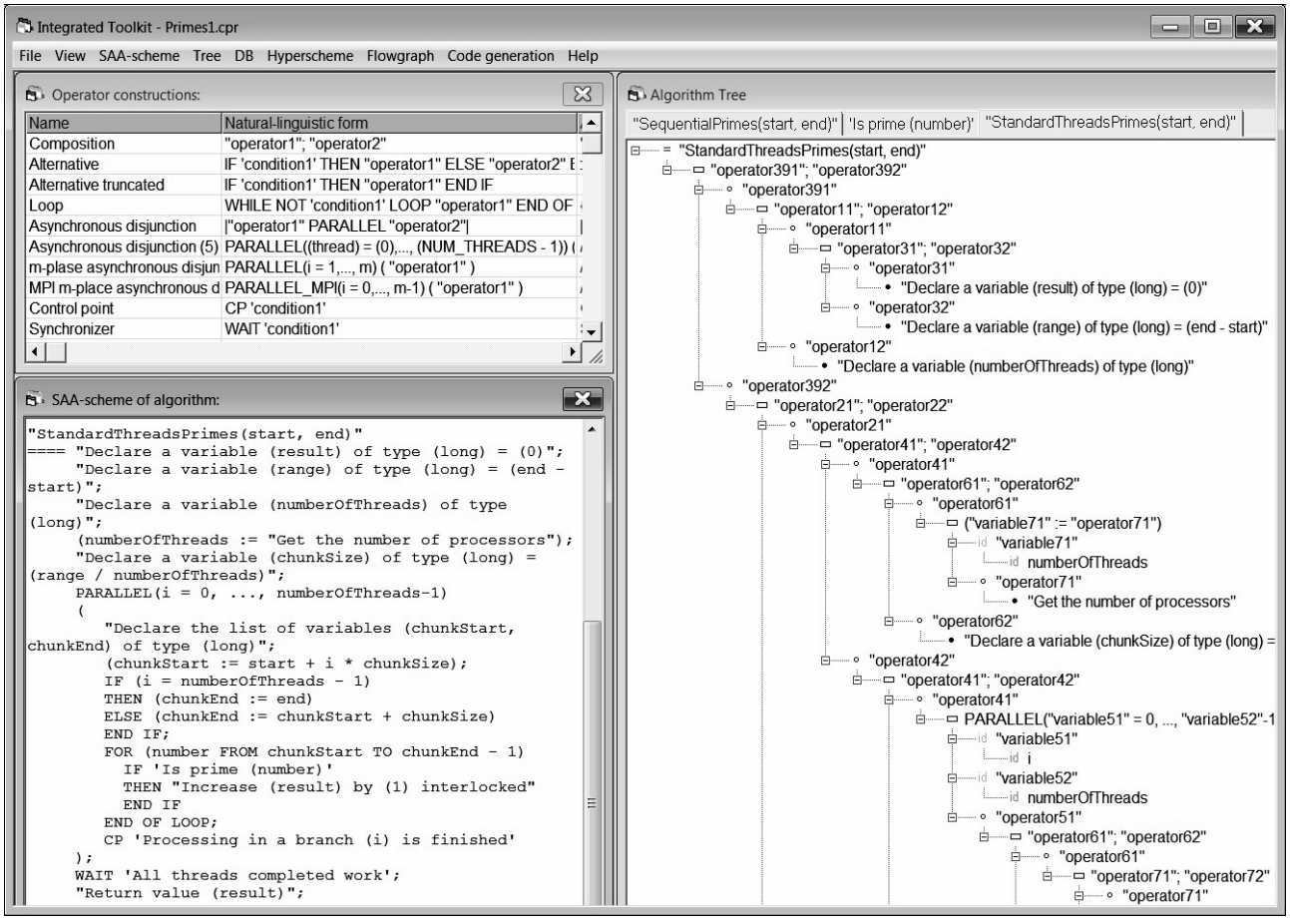

Рис. 1. Копія екрану інструментарію ІПС з побудованою схемою StandardThreadsPrimes

Сутність пулу потоків CLR полягає в наступному. При використанні різних коротких задач, що підлягають виконанню, можна заздалегідь створити набір потоків і потім відправляти відповідні запити, коли настає черга для їх виконання, таким чином, щоб кількість цих потоків автоматично збільшувалася із зростанням необхідності в цих потоках і зменшувалася при виникненні потреби в звільненні ресурсів. Для управління списком потоків в TPL передбачений клас ThreadPool, який в міру необхідності зменшує і збільшує кількість потоків у пулі до максимально допустимого значення. Значення максимально допустимої кількості потоків у пулі може змінюватися. У випадку двоядерного процесора воно за умовчанням становить 1023 робочих потоків і 1000 потоків введення-виведення [8].

Для підтримки використання пулу потоків у САА-М та систему ІПС введено конструкцію, яка відправляє певну операцію "*operator*" (задачу) у чергу на виконання:

```
QUEUE WORK ITEM
(
  "operator"
).
```

Вказана операція виконується, коли стає доступним потік із пулу потоків. Синхронізація потоків виконується за допомогою розглянутих у розділі 1 операцій контрольної точки та синхронізатора.

Приклад 3. САА-схема паралельного алгоритму знаходження кількості простих чисел із використанням пулу потоків є такою:

```
"ThreadPoolPrimes(start, end)"
==== "Declare a variable (result)
       of type (long) = (0)";
     "Declare a variable (chunkSize)
       of type (const long) = (100)";
     "Declare a variable (chunks) of
       type (long)";
     (chunks := (end – start) / chunkSize);
     FOR (i FROM 0 TO chunks – 1)
     "Declare the list of variables
       (chunkStart, chunkEnd)
       of type (long)";
     (chunkStart := start + i * chunkSize);
     IF (i = chunks – 1)
     THEN (chunkEnd := end)
     ELSE (chunkEnd := chunkStart +
           chunkSize)
     END IF;
     QUEUE WORK ITEM
     (
       FOR (number FROM chunkStart TO
            chunkEnd – 1)
         IF 'Is prime (number)'
         THEN "Increase (result) by (1)
               interlocked"
         END IF
       END OF LOOP;
       CP 'Thread (i) completed work'
     )
     END OF LOOP;
     WAIT 'All threads completed work';
     "Return value (result)".
```

У даному алгоритмі вхідний відрізок [*start, end*] поділяється на ділянки розміром *chunkSize* = 100, які обробляються задачами, що заносяться у чергу на виконання і виконуються потоками з пулу.

До алгебро-алгоритмічних засобів було включено також операцію паралельного циклу, яка відповідає конструкції Parallel.For бібліотеки TPL, і ітерації якої розподіляються між доступними у системі процесорами:

```
PARALLEL FOR (counter FROM
              start TO fin)
LOOP "operator"
END OF LOOP
```

Синхронізація потоків у рамках даної операції виконується автоматично.

Приклад 4. САА-схема паралельного алгоритму знаходження простих чисел, що використовує паралельний цикл, наведена далі.

```
"ParallelForPrimes(start, end)"
==== "Declare a variable (result)
       of type (long) = (0)";
     PARALLEL FOR (number FROM
                   start  TO end)
```

```
  IF 'Is prime (number)'
  THEN "Increase (result) by (1)
          interlocked"
  END IF
END OF LOOP;
"Return value (result)".
```

На основі розглянутих у прикладах 1–4 схем в системі ІПС виконана генерація програм мовою C# із використанням бібліотеки TPL. Зокрема, реалізації, що використовують пул потоків та паралельний цикл, є такими:

```
public static long ThreadPoolPrimes(
                    long start, long end)
{
  long result = 0;
  const long chunkSize = 100;
  long chunks;
  chunks = (end - start) / chunkSize;
  var completed = 0;
  var allDone = new
  ManualResetEvent(initialState:
                        false);
  for (long i = 0; i <= chunks - 1; i++)
  {
    long chunkStart, chunkEnd;
    chunkStart = start + i * chunkSize;
    if (i == chunks - 1)
    {
      chunkEnd = end;
    }
    else { chunkEnd = chunkStart +
           chunkSize; };
   ThreadPool.QueueUserWorkItem(
    _ =>
    {
      for (var number = chunkStart;
           number <= chunkEnd - 1;
           ++number)
      {
        if (IsPrime(number))
        {
          Interlocked.Increment(ref
                              result);
        };
      };
      if (Interlocked.Increment(
          ref completed) == chunks)
      {
        allDone.Set();
      }
    });
  };
  allDone.WaitOne();
  return result;
}

public static long ParallelForPrimes(
                    long start, long end)
{
  long result = 0;
  Parallel.For(start, end, number =>
  {
    if (IsPrime(number))
    {
      Interlocked.Increment(ref result);
    };
  });
  return result;
}
```

Результати виконання програм наведено у наступному розділі.

3. Результати експерименту

Проведено експеримент з виконання розроблених програм знаходження простих чисел SequentialPrimes, StandardThreadsPrimes, ThreadPoolPrimes та ParallelForPrimes на 4-ядерному процесорі Intel Core 2 Quad Q9300, 2.5 ГГц. На рис. 2 показано графік залежності часу виконання програм знаходження простих чисел на відрізку $[1, m]$ від значення m. Як видно з графіка, час виконання програм, що використовують TPL, а саме, ParallelForPrimes та ThreadPoolPrimes практично збігається, і є суттєво меншим ніж час виконання програми StandardThreadPrimes, яка застосовує стандартні засоби роботи з потоками.

Максимальне мультипроцесорне прискорення для StandardThreadPrimes становило 2.42 (ефективність використання ядер процесора 0.6), тоді як для програм, що використовують TPL – приблизно 4.0 (ефективність 1.0). Сумарний час виконання програми ThreadPoolPrimes (189.6 секунд) трохи перевищив час виконання ParallelForPrimes (187,9 секунд). Відмітимо також, що реалізація ParallelForPrimes є більш простою та компактною.

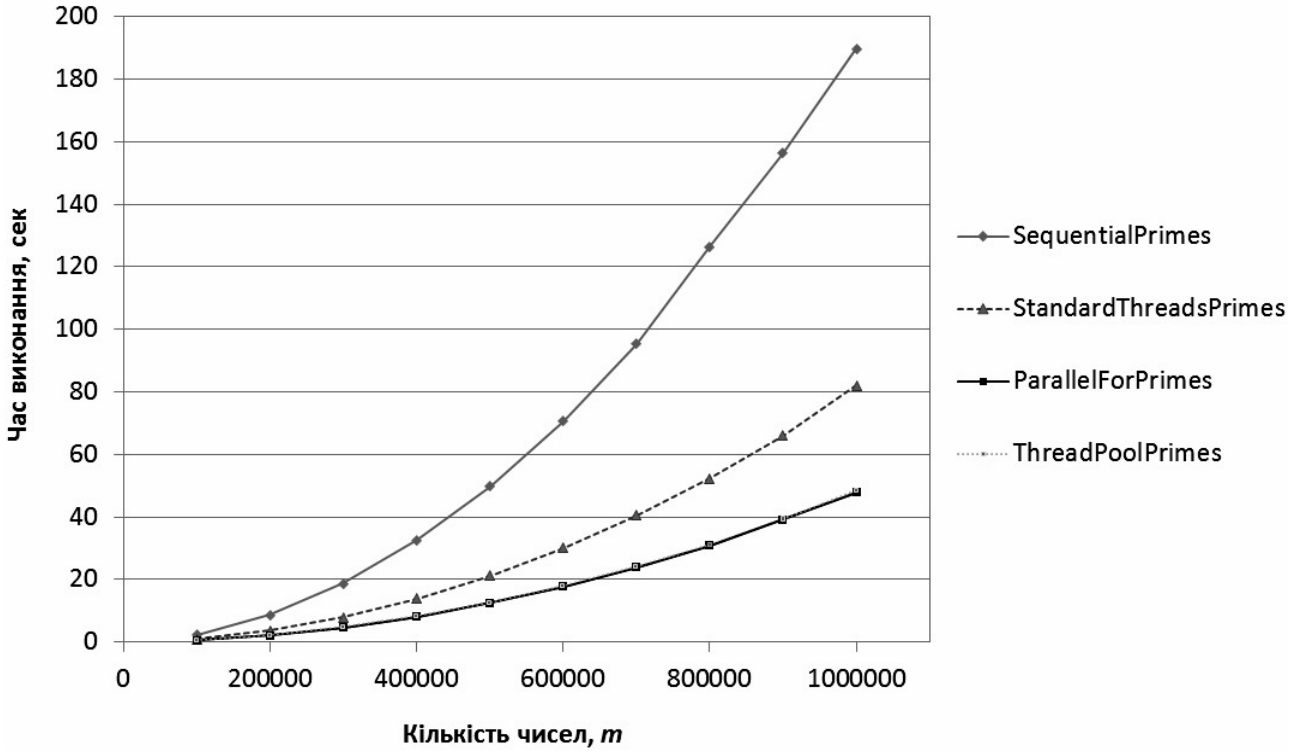

Рис. 2. Залежність часу виконання програм знаходження кількості простих чисел від розміру вхідних даних m

Висновки

Виконане налаштування алгебро-алгоритмічного інструментарію на формалізоване проектування та синтез паралельних програм мовою C#, що використовують засоби бібліотеки паралельних задач TPL. В основу пропонованого підходу покладені мова САА-схем, перевагою якої є простота в навчанні й використанні, та метод конструювання синтаксично правильних програм, що виключає можливість появи синтаксичних помилок у процесі проектування схем.

Проведено експеримент з виконання згенерованих за допомогою системи ІПС паралельних програм знаходження простих чисел на багатоядерному процесорі. Результати експерименту продемонстрували кращий ступінь розпаралелюваності обчислень для програм, що використовують TPL порівняно з програмою, що використовує стандартні засоби програмування потоків.

Література

1. Андон Ф.И., Дорошенко А.Е., Жереб К.А., Шевченко Р.С., Яценко Е.А. Методы алгебраического программирования. Формальные методы разработки параллельных программ. Киев: Наукова думка, 2017. 440 с.
2. Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А. Алгебро-алгоритмические модели и методы параллельного программирования. Киев: Академпериодика, 2007. 631 с.
3. Жереб К.А. Программный инструментарий, основанный на правилах, для автоматизации разработки приложений на платформе Microsoft .NET. *Управляющие системы и машин*ы. 2009. № 4. С. 51–59.
4. Дорошенко А.Е., Жереб К.А., Яценко Е.А. Об оценке сложности и координации вычислений в многопоточных программах. *Проблеми програмування*. 2007. № 2. С. 41–55.
5. Doroshenko A., Zhereb K., Yatsenko O. Developing and optimizing parallel programs with algebra-algorithmic and term rewriting tools. Communications in Computer and Information Science. Information and Communication Technologies in Education, Research,

and Industrial Applications. Springer, Heidelberg, 2013. Vol. 412. P. 70–92.
6. Task Parallel Library (TPL): сайт. URL: https://docs.microsoft.com/uk-ua/dotnet/standard/parallel-programming/task-parallel-library-tpl (дата звернення: 8.01.2020).
7. Microsoft Task Parallel Library* (TPL): сайт. URL: https://software.intel.com/en-us/node/811328 (дата звернення: 8.01.2020).
8. Васюткина И.А. Разработка клиент-серверных приложений на языке C#. Новосибирск: Изд-во НГТУ, 2016. 112 с.

References

1. Andon, P.I. et al. (2017) *Methods of algebraic programming. Formal methods of parallel program development*. Kyiv: Naukova dumka. (in Russian).
2. Andon, P.I. et al. (2007) *Algebra-algorithmic models and methods of parallel programming*. Kyiv: Akademperiodyka. (in Russian).
3. Zhereb K.A. The rule-based software toolkit for automation of development of applications on Microsoft.NET platform. *Upravl. Sistemy i Mashiny*. 2009. (4). P. 51–59. (in Russian).
4. Doroshenko A.Yu., Zhereb K.A., Yatsenko O.A. On complexity and coordination of computation in multithreaded programs. *Problems in programming*. 2007. (2). P. 41–55. (in Russian).
5. Doroshenko, A., Zhereb, K. & Yatsenko, O. (2013) Developing and optimizing parallel programs with algebra-algorithmic and term rewriting tools. In *Proc. 9th International Conference "ICT in Education, Research, and Industrial Applications" (ICTERI 2013), Revised Selected Papers*. Kherson, Ukraine, 19-22 June 2013. Berlin: Springer. 412. P. 70-92.
6. Docs.microsoft.com. *Task Parallel Library (TPL).* [online] Available from: https://docs.microsoft.com/uk-ua/dotnet/standard/parallel-programming/task-parallel-library-tpl [Accessed 8 Jan. 2020].
7. Software.intel.com. *Microsoft Task Parallel Library* (TPL).* [online] Available from: https://software.intel.com/en-us/node/811328 [Accessed 8 Jan. 2020].
8. Vasyutkina I.A. (2016) *Development of client-server applications in C# language*. Novosibirsk: Publishing House of Novosibirsk State Technical University. (in Russian).

Одержано 11.01.2020

***Про авторів*:**

Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук,
професор, завідувач відділу теорії комп'ютерних обчислень,
професор кафедри автоматики і управління в технічних системах НТУУ "КПІ імені Ігоря Сікорського".
Кількість наукових публікацій в українських виданнях – понад 150.
Кількість наукових публікацій в зарубіжних виданнях – понад 50.
Індекс Гірша – 5.
http://orcid.org/0000-0002-8435-1451,

Яценко Олена Анатоліївна,
кандидат фізико-математичних наук,
старший науковий співробітник.
Кількість наукових публікацій в українських виданнях – 44.
Кількість наукових публікацій в зарубіжних виданнях – 14.
Індекс Гірша – 2.
http://orcid.org/0000-0002-4700-6704.

Місце роботи авторів:

Інститут програмних систем
НАН України,
03187, м. Київ-187,
проспект Академіка Глушкова, 40.
Тел.: (044) 526 3559.

E-mail: doroshenkoanatoliy2@gmail.com,
oayat@ukr.net