УДК 004.451.45, 004.451.24, 519.687.1 PACS07.05.Bx; 07.05.Tp; 07.05.Kf; 02.70.-c; *01.50.hv*

Б.О. Онищенко

ДИРЕКТИВИ ТИПУ OPENMP ДЛЯ РОЗПАРАЛЕЛЮВАННЯ ЦИКЛІВ У МОВІ ПРОГРАМУВАННЯ АДА

Стаття містить опис реалізації директив технології OpenMP для розпаралелювання циклів засобами мови програмування Ада. Сучасні підходи до розробки ефективного програмного забезпечення передбачають активне використання технологій паралельного програмування. На даний момент існує і розробляється багато різних за своїми підходами технологій розробки паралельного програмного забезпечення. Особливу увагу слід приділяти тим підходам, які дозволяють розпаралелити вже готові послідовні алгоритми чи програми. До таких технологій відноситься OpenMP. Мова програмування Ада має власні засоби для розробки паралельних програм. Але їх використання у деяких випадках може бути ускладненими або недоцільним. Тому розробка підходів до розпаралелювання послідовних програм на мові Ада подібних до технології OpenMP є актуальною. Зокрема особливу увагу слід приділяти розпаралелюванню циклічних алгоритмів. У роботі наведені різні способи реалізації одного й того ж варіанту директиви та проведено аналіз їх ефективності й доцільності. Стаття містить приклади програмного коду.

Ключові слова*: технологія OpenMP, мова Ада, розпаралелювання циклів.*

Вступ

З огляду на те, що фактично кожен ПК зараз являє собою багатопроцесорну систему, тобто нарощуванні потужностей процесорів персональних комп'ютерів йдуть шляхом збільшення кількості їх ядер, у процесі проектування та розробки програмного забезпечення необхідно використовувати технології паралельного програмування. Зараз існує багато технологій розробки паралельних програм. На особливу увагу заслуговують засоби, що роблять можливим швидке «розпаралелення» вже розробленої послідовної програми шляхом додавання до її вихідного коду спеціальних інструкцій, за допомогою яких середовище компіляції сформує паралельний варіант коду та відповідний, паралельно працюючий виконуваний файл. До засобів такого типу належить технологія OpenMP [1, 2].

Технологія OpenMP, яка була розроблена для мов програмування C, C++ та Fortran, виявилась настільки зручною, що було зроблено декілька спроб адаптувати її для інших мов програмування. Зокрема в межах проекту PascalABC.NET реалізовано підтримку найбільш уживаних директив OpenMP [3] в середовище програмування PascalABC.NET. У роботах [4, 5] показано можливість адаптації технології OpenMP для мови програмування Ада.

Мета статті

Більшість програмних конструкцій, які можна представити як паралельно працюючі, являються конструкціями повторення, тобто циклами. Отже, метою даної статті є дослідження та опис роботи директив, подібних до директив стандарту OpenMP для розпаралелювання циклів, для мови програмування Ада.

Виділення проблеми, постановка задачі та формування цілей статті

Мова програмування Ада має власні засоби для розроблення паралельного програмного забезпечення. До таких засобів належать задачі (Tasks) та захищені модулі (Protected Units) [5, 7]. За допомогою зазначених мовних конструкцій можна розробити паралельно працюючу програму будь-якої складності. Однак, з іншого боку, використання вбудованих засобів розпаралелювання мови Ада може в окремих випадках надмірно ускладнити процес розробки програмного забезпечення та зробити текст програми недоступним для розуміння та аналізу. До таких випадків можна віднести програмування чисельних алгоритмів, алгоритмів, у яких проводиться обробка великих масивів даних тощо. Більшої зручності при розпаралелюванні таких алгоритмів надають засоби, подібні до технології OpenMP.

У роботі [4] показано, що кожній директиві OpenMP можна поставити у відповідність певну комбінацію мовних конструкцій мови Ада. У роботі [5] показана можливість введення директив, аналогічних до директив OpenMP, у мову програмування Ада, а також проаналізовано можливі способи заміни директив паралельними мовними конструкціями та представлена загальна послідовність дій під час переходу від послідовної Ада-програми з директивами OpenMP до паралельної Ада-програми. Дана робота є логічним продовженням роботи [5], у ній будуть проведені дослідження різних способів заміни директив для розпаралелювання циклів мовними конструкціями мови Ада з огляду на їх ефективність.

Опис директиви для розпаралелювання циклів

У роботі [5] обґрунтовано спосіб опису синтаксису директив OpenMP подібний до мови програмування Fortran. Запропонований синтаксис відрізняється від такого у мовах програмування C та C++ тим, що має початок та кінець, завдяки чому є більш логічним серед інших варіантів у випадку портування в мову Ада. Отже, опис директиви для розпаралелювання циклів у мові програмування Ада буде мати наступний вигляд:

```
--omp parallel for [опції]
for <заголовок циклу> loop
     <тіло циклу>
end loop
--omp end parallel for
```

На наступному кроці необхідно описати процедури заміни директив конструкціями мови Ада. Але в мові програмування є декілька можливостей опису паралельних фрагментів програмного коду. Тому потрібно дослідити ефективність застосування кожного з них.

Дослідження ефективності схем розпаралелювання циклів у мові Ада

Одним із засобів моделювання паралельних процесів є мережі Петрі [8]. Вперше описані Карлом Адамом Петрі в 1962 році, мережі Петрі є математичним апаратом для моделювання динамічних дискретних систем. Будь-яка мережа Петрі є дводольним орієнтованим мультиграфом, що складається з вершин двох типів: місця та переходу, з'єднаних дугами. Вершини місця можуть містити мітки, що здатні переміщуватися мережею. Подією в мережі Петрі називається спрацювання переходу, при якому мітки зі вхідних вершин місця переміщуються до вихідних.

Представимо за допомогою мереж Петрі процес утворення паралельних гілок при заміні директив OpenMP паралельними конструкціями цільової мови програмування. Основною проблемою паралельних програм є забезпечення ефективної міжпотокової

(міжзадачної) взаємодії, або синхронізації. На рівні задач у мові програмування Ада синхронізація можлива тільки за допомогою механізму рандеву, а захищені модулі значно розширюють можливості організації взаємодії між задачами. Отже, під час дослідження необхідно з'ясувати, які засоби синхронізації є більш ефективними і в яких випадках.

Згідно зі специфікацією OpenMP [1], кожен паралельний потік повинен отримати свій ідентифікаційний номер, починаючи з нуля, а у випадку розпаралелювання циклів, тобто використання директиви OpenMP `parallel for`, кожен потік повинен мати можливість отримання меж циклів за замовчуванням або згідно з опцією schedule. Отже, при розпаралелюванні циклів необхідно передбачити наявність щонайменше двох незалежних точок синхронізації у потоках.

Побудуємо мережі Петрі для фрагменту програми, в якому задачі отримують свої ідентифікаційні номери та межі циклу і використовують для цього різні засоби синхронізації. Мережа Петрі, що зображена на рисунку 1, відображає синхронізацію потоків із застосуванням механізму рандеву. Мережа Петрі, що зображена на рисунку 2, відображає синхронізацію потоків із застосуванням об'єктів захищених типів.

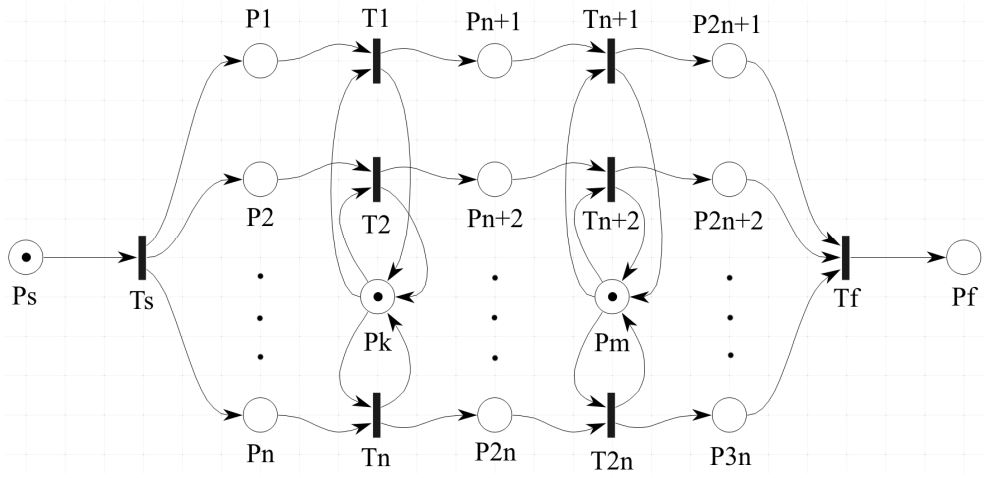

Рис. 1. Мережа Петрі, що моделює отримання потоками ідентифікаційного номеру та меж циклів за допомогою механізму рандеву.

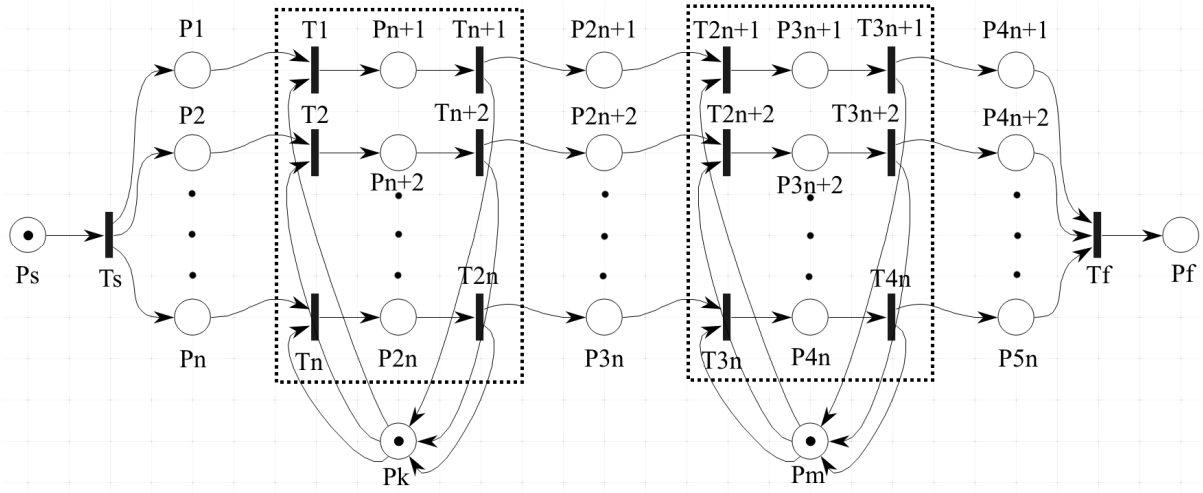

Рис. 2. Мережа Петрі, що моделює отримання потоками ідентифікаційного номеру та меж циклів за допомогою захищених модулів.

Як бачимо, спосіб отримання ідентифікатора та меж циклів однаковий для всіх потоків, тому відповідні вершини мережі Петрі, зображеної на рисунку 2, можна замінити двома макропереходами. З точки зору мови програмування Ада це означає, що підпрограми отримання ідентифікаторів та меж циклу можна винести з задач у захищений модуль. Отриману мережу Петрі зображено на рисунку 3.

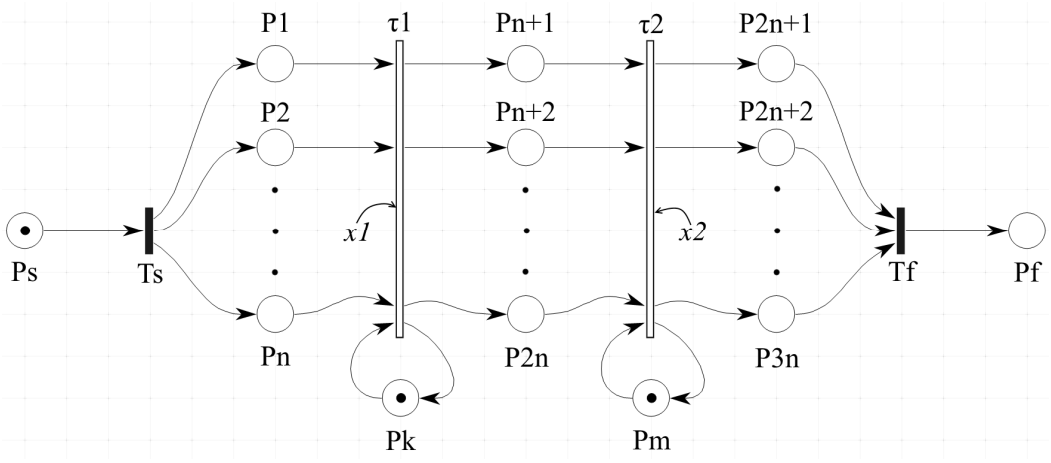

Рис. 3. Мережа Петрі з макропереходами, що моделює отримання потоками ідентифікаційного номеру та меж циклів за допомогою захищених модулів.

Мережа Петрі, зображена на рисунку 3, містить два управляючих вектори x_1 *і* x_2, які задають послідовність проходження мітки відповідно через блок отримання номеру та блок отримання меж циклів, що відносить її до класу управляючих мереж Петрі.

Отже, у процесі організації розпаралелювання за допомогою механізму рандеву для забезпечення коректної синхронізації паралельних ділянок програми необхідно залучати додаткові, синхронізуючі задачі, які будуть завжди використовувати процесорні ресурси. При використанні захищених модулів використовуються тільки ресурси пам'яті, а процесорні ресурси виділяються самими паралельними ділянками програми у випадку необхідності. Тобто використання захищених модулів є більш ефективним способом опису директив розпаралелювання циклів паралельними конструкціями мови Ада.

Заміна директив OpenMP до паралельних конструкцій мови Ада

Розглянемо тепер можливі способи заміни конкретних директив OpenMP на паралельні конструкції мови Ада. Нехай задано програму табуляції функції, що містить директиву OpenMP `parallel for` без параметрів:

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Functab_Directives is
      type TArray is array(-15..30) of Float;
      Arr: TArray;
      function F(X: in Float) return Float is
      begin
           return X * X - 5.0 * X + 4.0;
      end F;
begin
      --omp parallel for
      for I in -15..30 loop
         Arr(I) := F(Float(I));
```

```
            Put_Line("F(" & I'Img & ") = " & Integer(Arr(I))'Img);
        end loop;
        --omp end parallel for
end Functab_Directives;
```

У разі послідовного виконання програма обрахує та виведе значення даної квадратичної функції послідовно в усіх точках з –15 до 30.

Тепер перетворимо дану програму з використанням задач мови Ада за допомогою двох засобів міжзадачної взаємодії: спочатку механізму рандеву, а потім захищених модулів. Як було зазначено вище, при організації роботи паралельних потоків кожен з них повинен отримати свій ідентифікаційний номер, починаючи з нуля, а у випадку розпаралелювання циклів, тобто використання директиви OpenMP `parallel for`, кожен потік повинен мати можливість отримання меж циклів за замовчуванням або згідно з опцією `schedule`. Отже, ці особливості роботи технології OpenMP необхідно врахувати під час розробки паралельної версії програми:

```
with Ada.Text_IO; use Ada.Text_IO;
with System.Multiprocessors; use System.Multiprocessors;
procedure Functab_Tasks is
    type TArray is array(-15..30) of Float;
    Arr: TArray;
    function F(X: in Float) return Float is
    begin
      return X * X - 5.0 * X + 4.0;
    end F;
begin
    --omp parallel for
    declare
      OMP_NUM_THREADS: CPU := Number_Of_CPUs;
      Num_Threads: Integer := Integer(OMP_NUM_THREADS);
      Portion: Positive := 46 / Num_Threads;

      task type TThread is
           entry Set_ID(Arg: in Integer);
      end TThread;

      task body TThread is
           ID, From, To: Integer;
      begin
           accept Set_ID (Arg : in Integer) do
                ID := Arg;
           end Set_ID;
           From := -15 + (ID - 1) * Portion;
           if ID = Num_Threads then
                To := 30;
           else
                To := -15 + Portion - 1;
           end if;
           for I in From..To loop
               Arr(I) := F(Float(I));
               Put_Line("F(" & I'Img & ") = "&
                        Integer(Arr(I))'Img);
           end loop;
      end TThread;
```

```
        Threads: array(1..Num_Threads) of TThread;
 begin
        for I in Threads'Range loop
             Threads(I).Set_ID(I);
        end loop;
   end;
   --omp end parallel for
end Functab_Tasks;
```

У виведенні цієї програми можлива поява першими ітерацій останніх за індексом задач раніше, ніж перші за індексом задачі закінчать свою роботу. Це і є ознакою паралельної роботи даної програми. Також, як видно з лістингу, кожна задача отримує свій порядковий номер від головного потоку шляхом прийняття входу Set_ID.

Після отримання порядкового номера задачі на основі отриманого номера самостійно визначають, які саме ітерації циклу їм належить обробити. Такий підхід має ряд обмежень, а саме: при наявності опції `schedule` та відповідних параметрів у ній необхідно буде переписувати фрагмент коду, пов'язаний з розподілом ітерацій циклу між задачами, що, в свою чергу, ускладнює процес розробки узагальненого шаблону заміни директиви OpenMP `parallel for` паралельними конструкціями мови Ада, а при деяких варіантах параметрів у вищезгаданій опції взагалі неможливо зробити розподіл ітерацій в межах створених робочих потоків.

```
with Ada.Text_IO; use Ada.Text_IO;
with OpenMP_For_Ada; use OpenMP_For_Ada;
procedure Functab_Protected_Units is
    type TArray is array(-15..30) of Float;
    Arr: TArray;
    function F(X: in Float) return Float is
    begin
      return X ** 2 - 5.0 * X + 4.0;
    end F;
begin
    --omp parallel for
    OpenMP.Initialization(-15, 30);
    declare
      task type TThread;
      task body TThread is
           ID, From, To: Integer;
      begin
           OpenMP.Get_ID(ID);
           OpenMP.Get_Range_By_ID(ID, From, To);
           for I in From..To loop
                Arr(I) := F(Float(I));
                Put_Line("F(" & I'Img & ") = " & Arr(I)'Img);
           end loop;
      end TThread;
      Threads: array(1..OpenMP.Get_Num_Threads) of TThread;
    begin
      null;
    end;
    --omp end parallel for
end Functab_Protected_Units;
```

Як видно з порівняння наведених лістингів програм, використання захищених модулів як способу взаємодії задач покращує розуміння її вихідного коду програмістом, а також заощаджує обчислювальні ресурси, тому що захищений модуль займає лише виділену для нього ділянку оперативної пам'яті, а обчислювальні ресурси виділяються в межах тієї задачі, яка викликає описані в захищеному модулі процедури та функції.

Процедури присвоєння кожній задачі ідентифікатора, а також розподілу ітерацій початкового циклу було виделено у пакет, названий OpenMP_For_Ada:

```
with System.Multiprocessors; use System.Multiprocessors;
package OpenMP_For_Ada is
    protected OpenMP is
      function Get_Num_Threads return Integer;
      procedure Get_ID(New_ID: out Integer);
       procedure Get_Range_By_ID(ID: in Integer;
                                 From, To: out Integer);
      procedure Initialization(From, To: in Integer);
    private
      OMP_NUM_THREADS: Integer := Integer(Number_Of_CPUs);
      ID, First, Last: Integer;
    end OpenMP;
end OpenMP_For_Ada;
```

Функція Get_Num_Threads призначена для надання доступу на читання до захищеної змінної OMP_NUM_THREADS зовні пакету. Функція `Initialization` призначена для скидання генератора ідентифікаторів задач, а також передачі меж циклу в захищений модуль для подальшого розподілення його ітерацій між задачами. Функція Get_ID призначена для генерації послідовних чисел – ідентифікаторів задач, і працює подібно до SQL генераторів. Функція Get_Range_By_ID призначена для розподілення ітерацій циклу залежно від ідентифікатора задачі.

Перевагами використання такого пакету є те, що він займає тільки оперативну пам'ять і не витрачає обчислювальних ресурсів процесора, а також можливість його розширення іншими підпрограмами, наприклад, розподілу ітерацій між завданнями залежно від параметру опції `schedule`.

Усі опції директиви `parallel for` можна описати подібно до того, як це було зроблено у роботі [5]. Тому доцільним буде описати шаблон заміни директиви OpenMP `parallel for` паралельними конструкціями мови Ада.

Узагальнений шаблон заміни директиви OpenMP parallel for паралельними конструкціями мови Ада

Виходячи з наведених вище лістингів програм, можна спроектувати шаблон заміни директиви OpenMP `parallel for` відповідними паралельними конструкціями мови програмування Ада:

```
 ...
 with OpenMP_For_Ada; use OpenMP_For_Ada;
 ...
 --omp parallel for [private(<список>)[,]
       firstprivate(<список>)[,] lastprivate(<список>)[,]
       reduction(<оператор>: <список>)[,] schedule(<тип>[,
       <параметр>])]
OpenMP.Initialization(<cycle_first>, <cycle_last>);
```

```
declare
   protected Reduction is
      --reduction_setters | процедури-збирачі копій змінних
                           --редукції
      --reduction_getters | функції доступу до змінних редукції
   private
      --reduction_vars | оголошення змінних редукції
   end Reduction;
   protected body Reduction is
      --reduction_setters_bodies | тіла процедур-збирачів
      --reduction_getters_bodies | тіла функцій доступу
   end Reduction;
   task type TThread;
   task body TThread is
      ID, From, To: Integer;
      --local_thread_vars | оголошення приватних змінних
   begin
      OpenMP.Get_ID(ID);
       --init_firstprivate_vars | ініціалізація змінних зі
                                 --списку опції firstprivate
      OpenMP.Get_Range_By_ID(ID, From, To);
      for <cycle_iterator> in From..To loop
           --cycle_body_as_is | тіло циклу як є
      end loop;
      if To = <cycle_last> then
            --return_lastprivate_vars | повернення значень
                                    --змінних зі списку  lastprivate
      end if;
 end TThread;
 Threads: array(1..OpenMP.Get_Num_Threads) of TThread;
begin
 --return_reduction_vars | повернення результатів редукції
end;
--omp end parallel for
...
```

Основною перевагою даного шаблону є те, що всі інструкції, що утворюють необхідні для розпаралелювання циклу структури даних, розміщені в одному блоці. Це надає можливість за перший крок зробити більшу частину роботи по заміні директиви OpenMP `parallel for` відповідними паралельними конструкціями мови Ада. Решта ж роботи полягає у вставці змінної-ітератора, меж порцій циклу, а також приватних і редукційних змінних.

Іншою перевагою цього шаблону є спеціальні мітки для вставки таких його параметрів, як межі циклу чи приватні змінні. Варто зазначити, що обов'язкові параметри подано у шаблоні у кутових дужках, а необов'язкові – екрановано символами коментарів мови програмування Ада – дефісами. Це гарантує синтаксичну правильність результуючого коду: наприклад, якщо директива OpenMP `parallel for` не міститиме опції `lastprivate`, то в задачах буде лише порожній умовний блок, а не щось інше, синтаксично неправильне.

Існують два способи визначення останньої ітерації основного циклу для повернення значень `lastprivate`-змінних в основну програму. Перший спосіб полягає в перевірці рівності поточного значення ітератора номеру останньої ітерації основного циклу в кінці кожної ітерації циклу обробки порцій основного циклу. Цей спосіб має

алгоритмічну складність $\theta(n)$, де n – загальна кількість ітерацій. У свою чергу, другий спосіб полягає в перевірці рівності номерів останніх ітерацій поточної порції і основного циклу після обробки кожної порції. Другий спосіб має алгоритмічну складність $o(n)$, а отже, він є більш ефективним, ніж перший спосіб.

Висновки

Отже, у статті досліджено та описано ефективний спосіб реалізації директив OpenMP для розпаралелювання циклів у програмах на мові Ада. На наступному етапі необхідно розробити програму автоматичної модифікації програмного коду з директивами в програмний код, що містить паралельні конструкції мови Ада. Для реалізації такої програми доцільно було б використати специфікацію семантичного інтерфейсу мови Ада (Ada Semantic Interface Specification – ASIS) [8].

Список використаної літератури

1. OpenMP Application Program Interface Version 3.1. July 2011. [Електронний документ]. Режим доступу: http://www.openmp.org/mp-documents/OpenMP3.1.pdf. Перевірено: 18.07.2012.
2. Антонов А.С. Параллельное программирование с использованием технологии OpenMP. – М.: Изд-во МГУ, 2009. – 76 с.
3. Малеванный М.С. Реализация директив OpenMP для языка PascalABC. NET. Магистерская диссертация. – Ростов-на-Дону, 2011. – 52 с.
4. Stpiczyński P. Ada as a language for programming clusters of SMPs. – Annales UMCS Informatica AI 1, 2003. – P.P. 73–79.
5. Про можливість реалізації директив технології OpenMP для мови програмування Ада / О. Г.Лозова, Б. О. Онищенко, І. М. Сиволовський, І. М. Супруненко. // Східно-Європейський журнал передових технологій. – 2012. – №5/2(59). – С. 6–11.
6. Ada Reference Manual ISO/IEC 8652:2007(E) Ed. 3. [Електронний документ]. Режим доступу: http://www.adapower.com/rm95/index.html. Перевірено: 18.07.2012.
7. Гавва А. «Адское» программирование. Ada-95. Компилятор GNAT. [Електронний документ]. Режим доступу: http://ada-ru.org/V-0.4w/index.html. Перевірено: 18.07.2012.
8. Кузьмук В. В., Супруненко О. О. Модифицированные сети Петри и устройства моделирования параллельных процессов: Монография. – К.: Маклаут, 2010. – 252 с.
9. ASIS-for-GNAT Reference Manual [Електронний документ]. Режим доступу: http://docs.adacore.com/asis-docs/asis_rm.html. Перевірено: 18.07.2012.

References

1. OpenMP Application Program Interface Version 3.1. July 2011. Available at: http://www.openmp.org/mp-documents/OpenMP3.1.pdf (accessed 18 July 2012).
2. Antonov A.S. Parallel'noe programmirovanie s ispol'zovaniem technologii OpenMP (Parallel Programming with OpenMP technology). Moskov, Moscow State University Press, 2009. 76 p.
3. Maljovanyj M.S. Realisazija directiv OpenMP dlja jazyka PascalABC.NET. (Implementation of directives to OpenMP for languag PascalABC.NET.) Master's Thesis. Rostiv-na-Donu, 2011. 52 p.
4. Stpiczyński P. Ada as a language for programming clusters of SMPs. – Annales UMCS Informatica AI 1, 2003. – p.p. 73–79.

5. About possibility of implementation of directives of technology OpenMP for the Ada programming language / Lozova O.G., Onischenko B.O., Sivolovskij I.M., Suprunenko O.O. // Eastern European journal of enterprise technologies. – 2012. – № 5/2(59). – P. 6-11.
6. Ada Reference Manual ISO/IEC 8652:2007(E) Ed. 3. Available at: http://www.adapower.com/rm95/index.html (accessed 18 July 2012).
7. Gavva A. «Adskoe» programmirovanie. Ada-95. Kompiljator GNAT. ("Ada's" programming. Ada-95. Compiler GNAT) Available at: http://ada-ru.org/V-0.4w/index.html (accessed 18 July 2012).
8. Kuzmuk V.V., Suprunenko O.O. Modified Petri Nets and parallel process simulation devices: Monograph. – K.: McLaut, 2010. – 252 p.
9. ASIS-for-GNAT Reference Manual Available at: http://docs.adacore.com/asis-docs/asis_rm.html (accessed 18 July 2012).

Summary

B.O. Onyshchenko

OpenMP DIRECTIVES FOR PARALLELIZATION OF LOOPS IN ADA PROGRAMMING LANGUAGE

The article describes the implementation of the directives of OpenMP technology for parallelization of loops means Ada programming language. Modern approaches to the development of effective software provide extensive use of parallel programming techniques. At the moment, there is also developed a number of different approaches in their technology development of parallel software. Particular attention should be given to approaches that allow parallelize ready sequential algorithms or programs. These technologies include OpenMP. Ada programming language has its own tools for parallel program development. But their use, in some cases, it may be complicated or impractical. Therefore, the development of approaches to parallelization of sequential programs Ada like OpenMP technology is actuality. In particular, special attention should be given to cyclic parallelization algorithms. In the paper shows the different ways to implement the same version directive and the analysis of their effectiveness and appropriateness. This article contains software code examples.

Key words: *OpenMP technology, Ada language, loops parallelization*.

*Стаття надійшла _23_11_2015_*
*Прийнято до друку _27_11_2015_*