УДК 681.3

ФОРМАЛІЗОВАНЕ ПРОЕКТУВАННЯ ТА СИНТЕЗ ПАРАЛЕЛЬНОЇ ПРОГРАМИ ПОБУДОВИ ДІАГРАМИ ВОРОНОГО

К.А. Березовський, А.Ю. Дорошенко, О.А. Яценко

Інститут програмних систем НАН України,
03187, Київ, проспект Академіка Глушкова 40.
Тел.: 526 1538, e-mail: berezovskiy.kos@gmail.com,
doroshenkoanatoliy2@gmail.com, oayat@ukr.net

Розроблено високорівневі специфікації алгоритмів побудови діаграми Вороного у системах алгоритмічних алгебр. Виконано генерацію відповідного програмного коду цільовою мовою програмування із використанням розробленого інтегрованого інструментарію проектування та синтезу програм. Алгоритми подано у природно-лінгвістичній формі, особливостями якої є простота в навчанні та використанні, а також незалежність від мови програмування.

High-level specifications of algorithms, presented in systems of algorithmic algebra, for building Voronoi diagram are developed. Generation of corresponding code in a target programming language is implemented with usage of developed integrated toolkit for designing and synthesis of programs. The algorithms are presented in natural-linguistic form, the main feature of which is simplicity in learning and using, and also independence from programming language.

Вступ

Багатокутники Вороного вперше глибоко були вивчені українським математиком Г.Ф. Вороним [1], який використовував їх у роботі з квадратичних форм. Згадані діаграми мають багато предметних областей застосування [2]. Так, у фізиці сумісний вплив електричних і близькодіючих сил, для вивчення яких будуються складні діаграми Вороного, допомагає визначити структуру молекул. У біології використання діаграми, яка відображає картину розселення тварин і розподілу життєво важливих ресурсів, допомагає досліджувати та вивчати ефект перенаселення.

У роботі [3] була запропонована «ручна» розробка і паралельна реалізація алгоритму побудови діаграми Вороного на площині мовою C++, описано процес розробки алгоритму та програмування з використанням бібліотеки Intel Threading Building Blocks (TBB). У даній роботі здійснено формальне проектування та автоматизація генерації згаданої програми із використанням розробленого інтегрованого інструментарію конструювання та синтезу програм (ІПС) [4]. Інструментарій ґрунтується на основних поняттях алгебри алгоритміки [5, 6] і призначений для конструювання алгоритмів у вигляді схем (високорівневих формалізованих проектів), поданих у системах алгоритмічних алгебр В.М. Глушкова (САА). В основу побудови алгоритмів у системі ІПС покладено метод інтерактивного конструювання синтаксично правильних програм, який полягає у порівневому проектуванні схеми алгоритму зверху вниз із використанням заздалегідь розроблених елементарних компонентів (операцій) із виключенням можливості виникнення синтаксичних помилок [6]. У процесі конструювання використовуються три форми подання алгоритму: аналітична (формула в алгебрі алгоритмів), природно-лінгвістична (текстова) і графова (граф-схеми). У роботах [4, 7, 8] згаданий інструментарій було застосовано для синтезу багатопоточних програм за схемами асинхронних алгоритмів для виконання на багатоядерній архітектурі, розподілених MPI-програм для виконання на кластері та Грід-сервісів. При цьому розглядалося спільне використання ІПС і системи переписувальних правил для автоматизації трансформації паралельних програм. У даній роботі розроблене налаштування інтегрованого інструментарію на проектування та генерацію алгоритмів з нової предметної області – побудови діаграм Вороного.

Матеріал даної роботи складається з таких розділів: розділ 1 присвячено огляду основних функцій інтегрованого інструментарію, у розділі 2 виконане формалізоване проектування послідовного та паралельного алгоритмів побудови діаграми Вороного із використанням САА.

1. Інтегрований інструментарій проектування та синтезу програм

В основу розробленого інструментарію ІПС покладено поняття алгебри алгоритміки, яка є дворівневою системою. В основу верхнього рівня покладена теорія клонів (сімейств алгебр) [5]. На нижньому рівні алгебри алгоритміки здійснюється побудова цілком конкретної прикладної алгебри алгоритмів, орієнтованої на представлення алгоритмічних знань про обрану предметну область. Схема утворення прикладної алгебри базується на інтерпретації елементарних (базисних) операторних і предикатних змінних для алгебри алгоритмів, побудованої на верхньому рівні.

Прикладом алгебри алгоритмів є САА В.М. Глушкова [5, 6]. САА є двохосновною алгеброю $<U, B; \Omega>$, де U – множина операторів, B – множина логічних умов. Оператори є відображеннями інформаційної множини (ІМ) у себе, логічні умови є відображеннями множини ІМ у множину значень 3-значної логіки $E_3 = \{0, 1, \mu,\}$,

де 0 – хибність; 1 – істина; μ – невизначеність. Сигнатура операцій САА $\Omega = \Omega_1 \cup \Omega_2$ складається із системи Ω_1 логічних операцій, що приймають значення в множині *B*, і системи Ω_2 операцій, що приймають значення в множині операторів *U*. У САА < *U*, *B*; Ω > фіксується система утворюючих Σ, що є скінченною функціонально повною сукупністю операторів і логічних умов. За допомогою цієї сукупності із використанням суперпозиції операцій, що входять у Ω, породжуються довільні оператори і логічні умови з множин *U* і *B*.

На САА ґрунтується алгоритмічна мова САА/1 [5, 6], що використовується для проектування програм в ІПС. Згадана мова призначена для багаторівневого структурного проектування і документування послідовних і паралельних алгоритмів і програм. Перевагою її використання є можливість опису алгоритмів у природно-лінгвістичній формі, зручній для людини, що полегшує досягнення необхідної якості програм. Основними об'єктами мови САА/1 є абстракції операторів (перетворювачів даних) і умов (предикатів). Оператори й умови можуть бути базисними або складеними. Базисний оператор (умова) – це оператор (умова), що в САА-схемі вважається первинною атомарною абстракцією.

Складені умови будуються на основі базисних за допомогою логічних операцій САА:

- диз'юнкція: *'умова1' АБО 'умова2'*;
- кон'юнкція: *'умова1' І 'умова2'*;
- заперечення: *НЕ 'умова'*.

Складені оператори будуються з елементарних за допомогою операцій послідовного та паралельного виконання операторів:

- *"оператор1" ПОТІМ "оператор2"* (або *"оператор1"* ; *"оператор2"*) – послідовне виконання операторів;
- *ЯКЩО 'умова' ТО "оператор1" ІНАКШЕ "оператор2" КІНЕЦЬ ЯКЩО* – оператор розгалуження;
- *ПОКИ НЕ 'умова_закінчення_циклу' ЦИКЛ "оператор" КІНЕЦЬ ЦИКЛУ* – оператор циклу;
- (*"оператор1 ПАРАЛЕЛЬНО оператор2"*) – асинхронне виконання двох операторів (гілок);
- *ПАРАЛЕЛЬНО*(*i = 1, ..., n*)(*"оператор"*) – асинхронне виконання *n* операторів.

Ідентифікатори операторів в САА-схемах позначаються подвійними лапками, а умови – одинарними. Рівні в САА-схемах позначені лівими частинами рівностей, а структура кожного рівня конкретизується (в термінах САА) правою частиною відповідної рівності. Ліва частина рівності відокремлюється від правої ланцюжком символів "=".

ІПС призначений для проектування алгоритмів і генерації програм в цільових мовах програмування. Проектування алгоритмів у ІПС ґрунтується на методі діалогового конструювання синтаксично правильних програм [6] (ДСП-методі), що забезпечує синтаксичну правильність схем. При цьому він використовує згадані три різні форми подання алгоритмів у процесі їхнього проектування – природно-лінгвістичну (САА-схеми), алгебраїчну (регулярні схеми) і графу-схемну. ІПС підтримує генерацію послідовних і багатопоточних програм мовами Java та C++, а також MPI-програм мовою C++. До складу інтегрованого інструментарію входять такі компоненти:

- ДСП-конструктор, призначений для діалогового проектування синтаксично правильних схем послідовних і паралельних алгоритмів та генерації програм;
- редактор граф-схем;
- трансформатор, призначений для інтерактивного перетворення схем алгоритмів;
- генератор САА-схем на основі схем більш високого рівня – гіперсхем [5], які є узагальненням граматик структурного проектування;
- база даних, що містить опис конструкцій САА і базисних понять у трьох вищезгаданих формах, а також їх реалізації цільовою мовою програмування.

Основна ідея ДСП-конструктора (рис. 1) полягає у порівневому конструюванні схем алгоритмів зверху вниз із використанням конструкцій САА, що обираються користувачем зі списку. Процес побудови алгоритму полягає у суперпозиції згаданих конструкцій і представлений у вигляді дерева конструювання алгоритму. Обрані користувачем конструкції, а також операторні і логічні змінні, що входять в них, відображаються в дереві з подальшою деталізацією змінних. На кожному кроці проектування система в діалозі з користувачем надає на вибір лише ті конструкції, підстановка яких у текст, що формується, не порушує синтаксичну правильність схеми. У залежності від типу обраної змінної (операторного або логічного) система пропонує відповідний список операцій або базисних понять. Дерево конструювання алгоритму далі використовується для генерації тексту САА-схеми, регулярної схеми, граф-схеми і коду цільовою мовою програмування.

2. Формалізоване проектування алгоритмів побудови діаграми Вороного

Нагадаємо, що діаграма Вороного скінченної множини точок *S* на площині є таким розбиттям площини, за якої кожна область цього розбиття утворює множину точок, більш близьких до одного з елементів множини *S*, ніж до будь-якого іншого елемента множини. Виконаємо побудову [3] діаграми Вороного для множини точок площини із використанням підходу [9] на основі парадигми "розподіляй та володарюй" ("Divide and conquer").

Постановка задачі: Нехай маємо деяку множину P точок площини (рис. 2), які задовольняють вимогам для побудови діаграми Вороного, яку і слід побудувати (рис. 3).

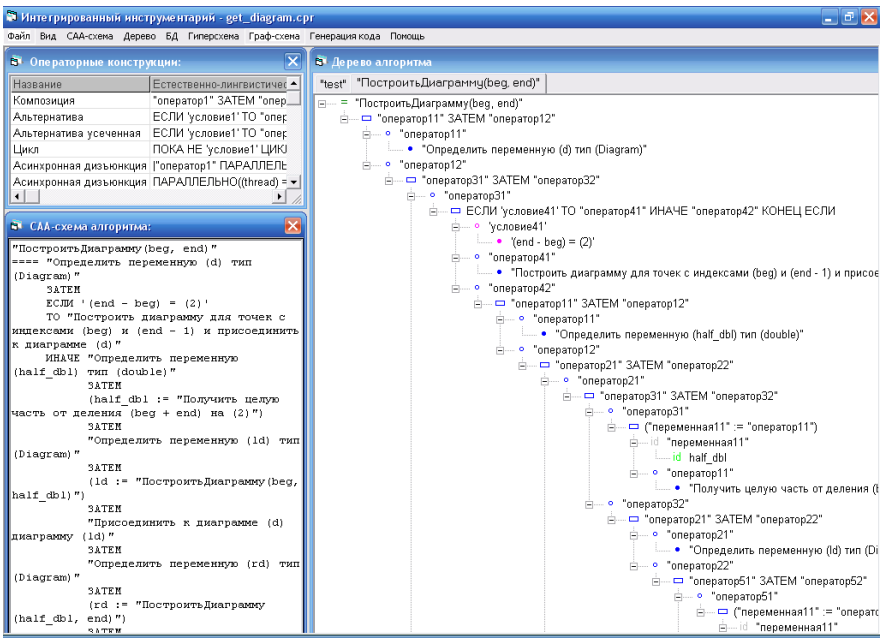

Рис. 1. Вікно ДСП-конструктора з прикладом алгоритму

Рис. 2. Вхідна множина P точок площини

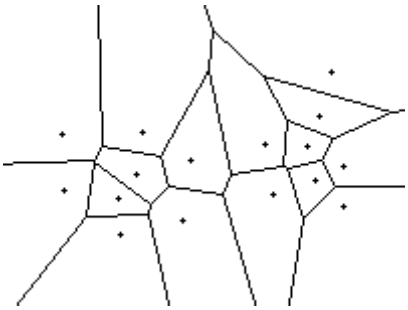

Рис. 3. Діаграма Вороного для множини P

Елементами множини P є пари $p(x, y)$, де $x, y \in N$. Впорядкуємо точки множини P за відношенням лінійного порядку R :

$$\forall \; p_1(x_1, y_1),\; p_2(x_2, y_2) \in P \qquad p_1 R p_2 \text{ якщо } \begin{bmatrix} x_1 < x_2 \\ або \\ x_1 = x_2, y_1 < y_2 \end{bmatrix}.$$

Для роботи з елементами P використаємо впорядкований масив p. Для представлення діаграми використаємо структуру даних *Diagram* = {*v*, *e*}, де *v* – масив вершин діаграми (vertices), *e*– масив ребер (edges). Згадана структура може бути подана в САА таким чином:

КЛАС Diagram
(
"Визначити масив (*v*) *типу* (*Point*)";
"Визначити масив (*e*) *типу* (*Edge*)";
...
)

Внаслідок генерації в інструментарії ІПС за наведеною конструкцією може бути отриманий такий фрагмент мовою програмування С++:

```
#include <vector>
class Diagram
{
    vector <Point> v;
    vector <Edge> e;
     ...
}
```

Кожне ребро представимо за допомогою шести компонент $\{v_1, v_2, e_1, e_2, p_1, p_2\}$, де v_1 – індекс першої (відповідно відношення R) вершини ребра (якщо вона існує) у масиві *v*: v_2 – індекс другої вершини ребра (якщо вона існує); e_1 – індекс точки у масиві *p*, що відповідає лівій грані діаграми (щодо даного ребра); e_2 – індекс точки у масиві *p*, що відповідає правій грані діаграми; p_1 – індекс суміжного відносно точки v_1 ребра, яке перетинається першим, при повороті даного ребра навколо точки v_1 проти годинникової стрілки; p_2 – індекс суміжного відносно точки v_2 ребра, яке перетинається першим, при повороті даного ребра навколо точки v_2 проти годинникової стрілки.

Для побудови САА-схеми алгоритму необхідні такі оператори:

- оператор "*Приєднати до діаграми* (*d1*) *діаграму* (*d2*)*"* для діаграм-операндів *d1* та *d2* повертає діаграму Вороного, що утворена шляхом конкатенації масивів операндів з оновленням індексів у компонентах v_1, v_2, p_1, p_2 (у мові С++ цей оператор реалізується за допомогою операції "+");
- оператор *"Побудувати діаграму для точок з індексами* (*beg*) *і* (*end-1*) *і приєднати до діаграми* (*d*)*"* повертає діаграму для двох точок з індексами *beg, end-1* та здійснює конкатенацію діаграми *d* з отриманою діаграмою (мовою С++ оператор реалізований із використанням відповідної процедури *getEdge*(*beg, end-1*));
- оператор *"Одержати цілу частину від ділення* (*beg + end*) *на* (*2*)*"* реалізований мовою С++ за допомогою процедури *floor* $(\frac{beg + end}{2})$;
- оператор *"Побудувати діаграму, що містить розділяючий ланцюг для діаграм* (*ld*) *і* (*rd*) *і приєднати її до діаграми* (*d*)*"* повертає діаграму Вороного з розділяючим ланцюгом для лівої (*ld*) і правої (*rd*) діаграм (мовою С++ реалізований за допомогою процедури *getDivChain*(*ld, rd*)).

Перелічені оператори були введені до бази даних інструментарію ІПС. Опис кожного оператора в БД включає його подання в природно-лінгвістичній та алгебраїчній формах, а також відповідне відображення цільовою мовою програмування (С++).

Із використанням введених операторів в інструментарії ІПС була сконструйована така САА-схема алгоритму побудови діаграми Вороного:

СХЕМА "ПобудуватиДіаграму(*beg, end*)*" =*

```
==== "Визначити змінну (d) типу (Diagram)";
     ЯКЩО "(end - beg) = (2)'
     ТО "Побудувати діаграму для точок з індексами (beg) і (end - 1) і приєднати до діаграми (d)"
     ІНАКШЕ
          "Визначити змінну (half) типу (double)";
          (half := "Одержати цілу частину від ділення (beg + end) на (2)");
          "Визначити змінну (ld) типу (Diagram)";
          (ld := "ПобудуватиДіаграму(beg, half)");
          "Приєднати до діаграми (d) діаграму (ld)";
          "Визначити змінну (rd) типу (Diagram)";
          (rd := "ПобудуватиДіаграму(half, end)");
          "Приєднати до діаграми (d) діаграму (rd)";
          "Побудувати діаграму, що містить розділяючий ланцюг для діаграм (ld) і (rd)
           і приєднати її до діаграми (d)"
     КІНЕЦЬ ЯКЩО;
     "Повернути значення (d)"
```

КІНЕЦЬ СХЕМИ "ПобудуватиДіаграму(*beg, end*)*"*

Текст більшості базисних елементів, що входять до схеми, містить у собі імена формальних параметрів, зазначені в дужках. Наприклад, текст базисного оператора

"Побудувати діаграму для точок з індексами (*i*) *і* (*j*) *і приєднати до діаграми* (*d*)*"*

містить три формальні параметри *i, j* та *d*. У процесі генерації програми значення фактичних параметрів підставляються замість відповідних формальних параметрів у тексті відображення мовою програмування С++. Текст реалізації мовою програмування для наведеного оператора має вигляд

%3 = %3 + getEdge(*%1*, *%2*);

де символ "%" є ознакою формального параметра, за яким вказується порядковий номер параметра в тексті базисного оператора.

Наприклад, внаслідок генерації в ІПС за текстом оператора *"Побудувати діаграму для точок з індексами* (*beg*) *і* (*end - 1*) *і приєднати до діаграми* (*d*)*"* буде отримано текст у програмі: *d = d + getEdge*(*beg, end - 1*).

Фрагмент програми мовою C++, що відповідає схемі алгоритму *"ПобудуватиДіаграму*(*beg*, *end*)*"*, має такий вигляд:

```
Diagram getDiagram(int beg, int end)
{
    Diagram d;
    if ((end-beg) == 2)
        d = d + getEdge(beg, end-1);
    else {
        double half = floor ((beg + end) / 2);
            // left diagram
            Diagram ld;
            ld = getDiagram(beg, half);
        d = d + ld;
        // right diagram
            Diagram rd;
        rd = getDiagram(half, end);
        d = d + rd;
        d = d + getDivChain(ld, rd);
    }
    return d;
}
```

Далі для знаходження розділяючого ланцюга (dividing chain) [10] доведеться розв`язувати задачу побудови „опуклої оболонки" ("Convex hull") [10] спеціального вигляду для точок площини.

Постановка задачі: Нехай маємо деяку множину P точок площини (рис. 4), деякі дві „крайні" [9, 10] точки p_{up}, $p_{down} \in P$, p_{up}, $p_{down} \in CH(P)$, причому ордината точки p_{down} має бути менша за ординату точки p_{up}, $y_{down} < y_{up}$. Через p_{down} і p_{up} проведемо пряму l. Слід побудувати частину опуклої оболонки $CH(P)$, що лежить по один бік від прямої l (рис. 5).

Рис. 4. Вхідна множина точок площини

Рис. 5. Опукла оболонка, що лежить з одного боку від прямої l

Використаємо відношення R для побудови рівняння прямої l ($a*x+b*y+c=0$), і за допомогою його можемо однозначно визначити положення точки $p \in P$ відносно прямої.

Для побудови САА-схеми модифікованого алгоритму QuickBall [9] необхідні такі оператори та предикати:

- предикат *'Список* (*sp*) *порожній'* приймає істинне значення, якщо список *sp* порожній, і хибне – в протилежному випадку (предикат реалізується в тексті програми мовою C++ за допомогою функції *isEmpty*(*sp*));
- оператор *"Заповнити список* (*sp*) *точками з множини* (*P*)*"* повертає список *sp*, заповнений елементами множини P (реалізується в тексті програми із використанням процедури *getCopy*(*P*));
- оператор *"Знайти точку* (*rp*) *зі списку* (*sp*), *що найбільш віддалена від прямої, яка проведена через точки* (p_{down}) *та* (p_{up})*"* повертає точку *rp* зі списку sp, що найбільш віддалена від прямої (яка проведена через точки p_{down}, p_{up}) і лежить з того боку, що описується маркером знаку *is_positive* (рис. 6) (цей оператор реалізується мовою C++ за допомогою процедури *getRemotePoint*(p_{up}, p_{down}, sp));

- оператор *"Отримати маркер знаку* (*is_positive*) *для верхньої опуклої оболонки* (p_{up} , p_{down})*"* повертає маркер знаку *is_positive*, що вказує на положення точок площини, які слід використовувати для побудови опуклої оболонки, відносно прямої, що проходить через точки p_{up} , p_{down} (рис. 7); цей оператор в тексті програми мовою C++ реалізований за допомогою процедури *getIsPositive*(p_{up} , p_{down});
- оператори *"Отримати список* (*up_sp*) *точок-кандидатів для верхньої опуклої оболонки* (p_{up} , *rp*, *sp* , *is_positive*)*"* та *"Отримати список* (*down_sp*) *точок-кандидатів для нижньої опуклої оболонки* (*rp*, p_{down}, *sp* , *is_positive*)*"* реалізуються мовою C++ із використанням процедури *getSidePoints*(p_{up} , p_{down}, *sp* , *is_positive*), яка повертає список точок, кожна з яких належить списку *sp* і лежить відносно прямої (p_{up} , p_{down}) відповідно маркеру знаку *is_positive*;
- оператор *"Видалити зі списку* (*sp*) *точку* (*p*)*"* повертає копію списку точок *sp*, у якій відсутня точка *p* (у тексті програми цьому оператору відповідає функція *erase*(*sp* , *p*));
- оператор *"Конкатенація списків*(*sp1*, *sp2*)*"* повертає конкатенацю списків точок *sp1* та *sp2* (відповідає функції C++ *concat*(*sp1*, *sp2*)).

Рис. 6. Точка, найбільш віддалена від прямої *l*

Рис. 7. Значення маркера знаку для точок площини, які лежать зліва та справа відносно прямої *l*

Із використанням введених предикатів та операторів була побудована така САА-схема модифікованого алгоритму *QuickBall*:

СХЕМА "quickBallModification(p_{up} , p_{down}, *sp* , *is_positive*)*" =*

==== *ЯКЩО 'Список* (*sp*) *порожній'*
ТО "Заповнити список (*sp*) *точками з множини* (*P*)*"*
КІНЕЦЬ ЯКЩО;
"Знайти точку (*rp*) *зі списку* (*sp*), *що найбільш віддалена від прямої,*
яка проведена через точки (p_{down}) *та* (p_{up})*"*;
ЯКЩО 'Точки (*rp*) *та* (p_{down}) *співпадають'*
ТО "Повернути значення (*sp*)*"*
ІНАКШЕ
"Отримати маркер знаку (*up_is_positive*) *для верхньої опуклої оболонки* (p_{up} , *rp*)*"*;
"Отримати список (*up_sp*) *точок-кандидатів для верхньої опуклої*
оболонки (p_{up} , *rp*, *sp* , *is_positive*)*"*;
"Видалити зі списку (*up_sp*) *точку* (p_{down})*"*;
"Отримати маркер знаку для нижньої опуклої оболонки (*rp*, p_{down})*"*;
"Отримати список (*down_sp*) *точок-кандидатів для нижньої опуклої*
оболонки (*rp*, p_{down}, *sp* , *is_positive*)*"*;
"Видалити зі списку (*down_sp*) *точку* (*rp*)*"*;
"up_convex_hull := quickBallModification (p_{up} , *rp*, *up_sp*, *up_is_positive*)*"*;
"down_convex_hull := quickBallModification (*rp*, p_{down}, *down_sp*,
down_is_positive)*"*;
"Повернути значення (*"Конкатенація списків*
(*up_convex_hull*, *down_convex_hull*)*"*) *"*
КІНЕЦЬ ЯКЩО

КІНЕЦЬ СХЕМИ "quickBallModification(p_{up} , p_{down}, *sp* , *is_positive*)*"*

Відповідне подання алгоритму мовою C++ має вигляд

```
quickBallModification ( p_up , p_down,  sp , is_positive)
{
    // якщо  sp  порожній, заповнимо його точками з P
    if (isEmpty(sp))
        sp = getCopy( P );
    // rp належить опуклій оболонці
    rp = getRemotePoint( p_up , p_down,  sp );
    if (rp ==  p_down){
        return sp;
    } else {
        // маркер знаку для верхньої опуклої оболонки
        up_is_positive =  getIsPositive( p_up , rp);
        // список точок-кандидатів для верхньої опуклої оболонки
        up_sp = getSidePoints( p_up , rp,  sp ,  is_positive);
        // видалення зайвої точки
        up_sp = erase(up_sp,  p_down);
        // маркер знаку для нижньої  опуклої  оболонки
        down_ is_positive =  getIsPositive(rp,  p_down);
        // список точок-кандидатів для нижньої опуклої оболонки
        down_sp = getSidePoints(rp,  p_down,  sp ,  is_positive);
        // видалення зайвої точки
        down_sp= erase(up_sp, rp);
        // конкатенація верхньої та нижньої опуклої оболонок
        return concat(quickBallModification( p_up , rp, up_sp, up_is_positive),
        quickBallModification(rp,  p_down,  down_sp, down_is_positive));
}}
```

Виконаємо розпаралелення наведеного на початку розділу алгоритму *"ПобудуватиДіаграму*(*beg*, *end*)*"*. У паралельному алгоритмі побудова "лівої" та "правої" діаграм (*ld* та *rd*) здійснюється рекурсивно і паралельно. При цьому, якщо кількість вхідних точок невелика, тобто є меншою певного граничного значення *cutOff*, то виконується послідовний алгоритм, а інакше – паралельний (оскільки виконання послідовної програми при малій кількості точок є швидшим за виконання паралельної). Для побудови паралельної САА-схеми необхідно виконати такі перетворення послідовного алгоритму:

Трансформація 1:

'(*end - beg*) = (*2*)' ⇒ '(*end - beg*) < *cutOff*'

Трансформація 2:

"Побудувати діаграму для точок з індексами (*beg*) *і* (*end - 1*) *і приєднати до діаграми* (*d*)*"* ⇒
(*d* := *"ПобудуватиДіаграму*(*beg*, *end*)*"*)

Трансформація 3:

(*ld* := *"ПобудуватиДіаграму*(*beg*, *half*)*"*); ⇒ ((*ld* := *"ПобудуватиДіаграму*(*beg*, *half*)*"*)
(*rd* := *"ПобудуватиДіаграму*(*half*, *end*)*"*); ПАРАЛЕЛЬНО
(*rd* := *"ПобудуватиДіаграму*(*half*, *end*)*"*))

У тексті наведених трансформацій символ "⇒" означає заміну в алгоритмі лівого виразу на правий. Внаслідок застосування трансформацій одержимо таку паралельну САА-схему:

СХЕМА "ПобудуватиДіаграмуПаралельно(*beg*, *end*)*" =*

==== *"Визначити змінну* (*d*) *типу* (*Diagram*)*"*;
ЯКЩО '(*end - beg*) < *cutOff*'
ТО (*d* := *"ПобудуватиДіаграму*(*beg*, *end*)*"*)
ІНАКШЕ

"Визначити змінну (*half*) *типу* (*double*)*"*;
(*half* := *"Одержати цілу частину від ділення* (*beg* + *end*) *на* (*2*)*"*);
"Визначити змінну (*ld*) *типу* (*Diagram*)*"*;
"Визначити змінну (*rd*) *типу* (*Diagram*)*"*;
((*ld* := *"ПобудуватиДіаграму*(*beg*, *half*)*"*)
ПАРАЛЕЛЬНО
(*rd* := *"ПобудуватиДіаграму*(*half*, *end*)*"*));
"Приєднати до діаграми (*d*) *діаграму* (*ld*)*"*;
"Приєднати до діаграми (*d*) *діаграму* (*rd*)*"*;
"Побудувати діаграму, що містить розділяючий ланцюг для діаграм (*ld*) *і* (*rd*)
і приєднати її до діаграми (*d*)*"*
КІНЕЦЬ ЯКЩО;
"Повернути значення (*d*)*"*

КІНЕЦЬ СХЕМИ "ПобудуватиДіаграмуПаралельно(*beg*, *end*)*"*

За наведеними в даному розділі САА-схемами алгоритмів в інтегрованому інструментарії була виконана генерація послідовної та багатопоточної програм мовою C++. Паралельна програма реалізована за допомогою бібліотеки TBB. При цьому програмна реалізація операції паралельного виконання операторів, наведена у схемі *"ПобудуватиДіаграмуПаралельно*(*beg*, *end*)*"*, розроблена із використанням поняття "логічної задачі" (клас `tbb::task`) [11].

Висновки

Розроблено алгеброалгоритмічні специфікації послідовного та паралельного алгоритмів побудови діаграми Вороного. Виконано генерацію відповідного програмного коду мовою C++ із використанням розробленого інтегрованого інструментарію проектування та синтезу програм. Алгоритми подано в природно-лінгвістичній формі (мові САА/1), особливостями якої є простота в навчанні та використанні, а також незалежність від мови програмування і можливість автоматизованого перекладу в довільну мову для забезпечення відповідності алгоритмів та асоційованих з ними програм.

1. *Вороний Георгій Феодосійович*. – http://uk.wikipedia.org/wiki/Вороний_Георгій_Феодосійович.
2. *Анісімов А.В., Терещенко В.М., Кравченко І.В*. Основні алгоритми обчислювальної геометрії. – http://cg.unicyb.kiev.ua.
3. *Березовський К.А*. Дослідження паралельного алгоритму побудови діаграми Вороного на площині // Проблеми програмування. – 2009. – № 1. – С. 28–35.
4. *Дорошенко А.Е., Жереб К.А., Яценко Е.А*. Формализованное проектирование эффективных многопоточных программ // Проблеми програмування. – 2007. – № 1. – С. 17–30.
5. *Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А*. Алгеброалгоритмические модели и методы параллельного программирования. – Киев: Академпериодика, 2007. – 631 с.
6. *Цейтлин Г.Е*. Введение в алгоритмику. – Киев: Сфера, 1998. – 311 с.
7. *Дорошенко А.Е., Жереб К.А., Яценко Е.А*. Средства синтеза параллельных MPI-программ // Проблеми програмування. – 2008. – № 2–3. – С. 595–604.
8. *Дорошенко А.Е., Яценко Е.А*. Средства сервисно-ориентированного программирования параллельных программ // Проблеми програмування. – 2009. – № 2. – С. 12–21.
9. *Препарата Ф., Шеймос М*. Вычислительная геометрия: Введение / Пер. с англ. – М.: Мир, 1989. – 478 с.
10. *Cormen T.H., Leiserson C.E., Rivest R.L., Stein C*. Introduction to Algorithms, Second Edition. – MIT Press and McGraw-Hill, 2001. – 1184 p.
11. *Intel* Threading Building Blocks 2.2. – http://www.threadingbuildingblocks.org/