УДК 681.03

А.Ю. Дорошенко, М.В. Котюк, С.С. Ніколаєв

ПРОГРАМНА ПЛАТФОРМА ДЛЯ НАУКОВИХ ДОСЛІДЖЕНЬ

Описано нову програмну платформу для розв'язання задач, які виникають у процесі наукових досліджень, що характеризується високою продуктивністю та захищеністю генерованого коду, а також містить засоби компонентно-орієнтованого та евристичного програмування.

Вступ

В останні роки спостерігається тенденція зростання популярності двох домінуючих сучасних програмних платформ .NET та J2EE [1, 2], а також все меншого використання розробниками «одиночних» технологій та мов програмування, таких як Delphi, C++Builder, MS Visual Basic 6 та ін. Причина популярності платформ полягає в тому, що створювати програми з їх допомогою значно ефективніше, а результати роботи отримують велику кількість переваг за рахунок самої платформи: сумісність, можливість застосування на ЕОМ будь-якої архітектури, можливість повторного використання, висока надійність, можливість простої та стандартизованої взаємодії створених різними програмістами компонентів. Проте слід зазначити низку вад використання таких програмних платформ як .NET чи Java.

По-перше, створювані з допомогою цих платформ програми мають невисоку продуктивність внаслідок використання технології віртуальних машин (ВМ), що передбачає компіляцію та інтерпретацію проміжного коду. Спеціальні процесори з підтримкою апаратної інтерпретації проміжних кодів поширення не набули, а використання об'єктно-орієнтованого програмування (ООП) як основного методу розробки та не дуже економного щодо споживання основної пам'яті «збирання сміття» теж не сприяє високим показникам продуктивності таких програм.

По-друге, в сучасних версіях платформ .NET та Java досі не в повному обсязі реалізовані технології інформаційної безпеки та захисту інтелектуальної власності (хоча технічно це можливо).

По-третє, ці платформи вимагають використання програмістом однієї концепції, однієї базової бібліотеки та однієї мови програмування (проголошена «багатомовність» .NET є умовною, бо всі мови .NET насправді мають Java-подібну семантику, хоча і різний синтаксис, окремо в деякій мірі стоять лише CIL та Managed Extensions for C++). І тому в цьому випадку виключається можливість використання цих програмних платформ для досліджень в галузі програмування та створення *справді нових* мов та методів програмування.

Разом ці фактори вказують на те, що використання платформ Java та .NET може різко звужувати і обмежувати можливості для наукових досліджень, зокрема у галузі програмування. У програмних системах, що виконують наукові обчислення, критичними є саме продуктивність обчислень, захист інформації та можливість створення нових методів і засобів опису, виконання та моделювання алгоритмів. З появою мультипроцесорних систем та мультиядерних процесорів з'явилося ще одне питання – як максимально ефективно використати технології паралельного виконання програмного коду. Проте і тут ні Java, ні .NET ще не запропонували розв'язання цих проблем, тому науковці досі використовують такі давно відомі засоби, як C/C++, Fortran чи Lisp.

Програмна платформа МИША – це нова програмна платформа, що пропонує вирішення вищеназваних та інші технологічних проблем шляхом оптимального набору засобів для використання в наукових дослідженнях з програмування.

Перша версія МИШІ була створена в 2005 році М.В. Котюком як компонент створеної експериментальної навчальної системи, призначений для написання інтелектуальних алгоритмів – аналізу тексту для автоматичного генерування тестових

завдань, аналізу вимови та розпізнавання символів тощо. Після цього було виконано модернізацію системи, яка значно розширила спектр застосування МИШІ у галузі наукових обчислень. МИША М5 включала в себе велику кількість цікавих технологій, у тому числі динамічну кодогенерацію в момент виконання, можливість виконання програми на графічному акселераторі, автоматизовану систему розгортання ПЗ на кластерному обчислювачі та технологію інтелектуальної оптимізації коду. У 2005-2007 роках МИША була успішно представлена на конкурси-захисти науково-дослідних робіт МАН. У 2006 році до розробки компонентів системи долучився С.С. Ніколаєв, який створив засоби обробки файлів, мультимедійні компоненти, систему захисту інформації та розробив мультиасемблерну мову запису евристик.

Склад та особливості платформи МИША

МИША – це багатомовна кросплатформна технологія, що дає змогу інтегрувати компоненти, створені з використанням різних мов програмування з різними парадигмами (імперативна та декларативна), для реалізації складних науково-дослідницьких проектів. Платформа складається з таких компонент:

- операційної системи реального часу МИША ОС 1.0, яка може використовуватися для виконання програми на базі платформи МИША;
- дистрибутивів МИШІ для ОС Windows NT, Linux, Macintosh, Solaris, DOS, Symbian, Windows CE;
- засобів виконання МИША-програм на графічних акселераторах компанії AMD;
- технології Universal Binary, що забезпечує виконання програми у оригінальному машинному коді, без використання інтерпретації/JIT-компіляції проміжного коду;
- технології захисту авторського права Tiger II;
- системного антивіруса/антишпигуна “Сотник”, який гарантує відсутність шкідливого програмного коду у МИША-програмах;
- універсальної «прозорої» системи взаємодії з операційними системами різних типів;
- універсальної системи типів даних, оптимізованої для наукових досліджень;
- високорівневої проміжної універсальної мови програмування, в яку перекладаються всі програми на базі МИША-сумісних мов програмування;
- технології розпаралелювання та оптимізації програм для багатопотокового виконання, засоби розробки програм та їх розгортання на кластерних системах.
- система управління пам'яттю на основі використання символьних імен;
- автоматичне управління «купою» на базі миттєвого «збирання сміття», що гарантує мінімальний обсяг використання пам'яті та відсутність об'єктів без посилань в будь-який момент часу, тобто фіналізація об'єктів у МИШІ – це детермінований процес (на відміну від .NET та Java).

Ідеологія створення програмних систем на базі платформи МИША отримала назву алгоритмічно-орієнтованого програмування (АОП), яка має багато спільного з алгеброалгоритмічним підходом до проектування алгоритмів [3]. АОП дає змогу використати в кожній ситуації оптимальний підхід до проектування системи чи окремого алгоритму. Так, наприклад, можна поєднувати для використання традиційну структурну модель Паскаля, об'єктно-орієнтовну модель в стилі .NET, технологію структурування даних МИШІ, а також евристичні методи та декларативне програмування.

Важливою перевагою платформи МИША є високоефективна реалізація бібліотек та транслятора. Так, бібліотеки для ЕОМ архітектур x86, AMD FireStream та МП реалізовані мовою асемблера.

Транслятор для платформи МИША є триступеневою системою, що забезпечує високий рівень технологічної сумісності й можливість створення нових трансляторів для багатьох мов програмування. Технологічно на всіх рівнях транслятор реалізований як інтелектуальна нейронна мережа, із використанням динамічно генерованих процедур та евристик. Транслятор пропо-

нує засоби профілювання та інтелектуальної оптимізації програмного коду на рівні вхідних текстів і асемблерного коду.

Важливою рисою транслятора є можливість програмних систем динамічно змінювати власний високорівневий код, що пропонує широкі можливості зі створення націленої компіляції та інтелектуальних програмних комплексів [4].

Власне поточна (десята) версія транслятора МИША є гарною демонстрацією можливостей платформи та самої мови програмування МИША М10, яка є основною мовою платформи, що являє собою високорівневу мову програмування, в якій підтримуються різні способи опису алгоритмів.

При створенні мови програмування МИША за основу було взято мову Паскаль. В цій мові програмування поєднуються важливі якості розвинутої функціональності та простоти в освоєнні. Мову збагачено конструкціями, які дозволяють якісніше описувати алгоритми та забезпечують позбавлення від поширених техніко-алгоритмічних помилок.

Значна частина елементів мови програмування запозичена з мов програмування Алгол, Ліспа та С. В МИШІ чіткість та однозначність семантики Алголу-60 поєднується із багатьма елементами С, що дає змогу писати компактніший та зрозуміліший код. Наявна також підтримка об'єктно-орієнтованого програмування. МИША М10 дозволяє створювати програми з допомогою декларативних засобів.

Синтаксис мови програмування стандартизовано. Це прискорює освоєння мови програмування: досить лише осмислити головні властивості та закономірності. Мова програмування дозволяє максимально ефективно оптимізувати спільну роботу в певному проекті та повністю використовувати наявні обчислювальні ресурси. Створення програми для паралельного виконання нічим не відрізняється від створення звичайної програми. Система програмування для МИШІ – Омега 2007 – значно збільшує швидкість створення програми чи алгоритму. Пропонується багато засобів, що спрощують пошук помилок у програмі. Синтаксис мови МИША зовні нагадує C++, а зсередини – Паскаль. Багато елементів є звичними для досвідчених програмістів, і це допомагає у вивченні мови МИША М10.

Програма мовою МИША зберігається в єдиному файлі з розширенням імені .cpp (мова C++). Розширення .cpp використане для того, щоб в одній системі Омега 2007 об'єднати три мови (C, C++ та МИША) з єдиним розширенням імені файлів для спрощення реалізації транслятора. Незручностей у роботі в цьому контексті не виникає, адже система Омега 2007 автоматично розпізнає мову (МИША або C++) і завантажує відповідні програмні засоби.

МИША підтримує препроцесор, що є вкороченою версією препроцесора C/C++: підтримується умовна компіляція та включення файлів.

Файл складається з двох частин – зони оголошень та зони опису головного тіла програми. В зоні оголошень описуються підпрограми, модулі, типи. Головне тіло програми – це "дорожня карта" програми, з якого починається і яким закінчується її виконання.

Приклад програми мовою МИША: (приклад 1).

Текст, що не аналізується транслятором – це коментарі. В МИШІ є 3 типи коментарів – однорядкові (//текст коментаря), багаторядкові (/*текст коментаря*/), та документаційні директиви – вказівки, що аналізуються середовищем програмування, але ігноруються транслятором.

Конструкції мови МИША поділяються на 2 групи – виконувані (вирази) та невиконувані (оголошення). Стиль запису виразів у мові МИША майже ідентичний запису виразів у мові С, але має багато особливостей, викликаних характеристиками платформи. Так, МИША подібно до декларативних мов використовує передачу даних за значенням, що дає змогу легко реалізувати мережеву взаємодію програмних компонентів та інші види обчислень на архітектурах із розподіленою пам'яттю. Водночас наявна **var**-специфікація для передачі аргументів до підпрограм. Приклад 1 програми мовою МИША.

```
//Вказівка на тип мови
// МТ="Миша-Тип"
/**МТ::MAINFILE*/
#include "mcl.hpp"

//Зона оголошень
program
{
     //операторі головного тіла програми
     Console.PETC();
     // очікування введення інформації
     return EXIT_SUCCESS
     //подання інформації про вдале завершення програми
}
```

Звідси, в МИШІ немає можливості працювати із вказівниками. Але платформа пропонує технологію маршалінгу об'єктів за посиланням (**reference**). Для цього використовується надійний та економний протокол передачі даних у бінарному форматі.

Так, у мові МИША вірний такий алгоритм:

```
ref<IEnumerable> g5=i[3];
ref<Object(g5)().SetTag(10).
```

На відміну від системи .NET Remoting, МИША не передбачає різних типів активації, створення та лізингу об'єктів, адже використовувана в МИШІ сервіс-орієнтована архітектура, на відміну від клієнт-серверної, чітко структурує способи використання переданих за посиланням об'єктів.

Система типів мови МИША

Система типів мови програмування МИША традиційна і багато в чому аналогічна системі типів у Паскалі. Відмінні назви деяких типів, а також наявні деякі типи, відсутні в мові Паскаль.

У наукових обчисленнях потрібні як зручні методи опису структури даних, так і зручні методи опису списків та інших видів структур однотипних даних. МИША пропонує три види табличних величин – рядки, масиви та деревоподібні структури, а також можливість опису структур даних і прив'язаних до них підпрограм за допомогою записів (record). Водночас деревоподібні структури виконують майже всі функції записів, на додачу пропонуючи властивість, якою наскрізь пронизана ідеологія МИШІ – динамічність.

Бібліотеки МИШІ – це дуже потужний інструмент розробника. В них знаходяться близько тисячі типів даних та дев'яти тисяч підпрограм, згрупованих за тематичним застосуванням. Типи і функції МИШІ – це не лише загальні, а й спеціалізовані. Так, окрім типів для роботи з натуральними дробами, підтримується бібліотека для фізичного моделювання, бібліотека з функцією хімічного калькулятора тощо.

Незважаючи на те, що МИША передбачає створення однофайлових програм, основним методом розробки є компоненто-орієнтоване програмування. Мова МИША передбачає створення різних типів незалежних компонентів: структур даних із базовими алгоритмами, що можуть бути повторно використані та модулів коду із утилітарними алгоритмами.

Записи в МИШІ – часткові аналоги класів у об'єктно-орієнтованих мовах програмування. Так, записи підтримують успадкування типів та реалізацію інтерфейсів (інтерфейс – це інформаційний тип даних, що забезпечує алгоритмічний поліморфізм МИША-алгоритмів, схожий на інтерфейси в мові Паскаль). Табличні величини – ще одна особливість мови МИША, що дозволяє зробити значний крок уперед в галузі проектування програмного забезпечення. Вони забезпечують швидку, зрозумілу та автоматизовану обробку та гарантують надійність і ефективність програми.

Назва типу в МИШІ	*Назва типу в Паскалі*	*Опис, обсяг необхідного простору в ОЗП 64-розрядної ПЕОМ під управлінням ОС МИША*
bool	boolean	Логічне значення, 8 змінних на байт (МИША), 1 байт (Паскаль)
byte	byte	Байт, 1 байт
char	char	Символ, 1 байт
PChar	PChar	NT-рядок (завершується символом 0)
int	integer	Ціле число з розрядністю використовуваної комп'ютерної системи
int64	int64	64-розрядне ціле число
real	real	64-розрядне дійсне число
extended	extended	128-розрядне дійсне число
str	**string**	Символьний рядок, 8 байт+довжина рядка
var	variant	Варіантний контейнер для будь-якого типу даних – від 52 байтів
anytype	pointer, **dynamic array**	Масив, що реалізується через вказівник на область динамічної пам'яті – 4 байти
sar	**array**	Стандартний Паскаль-масив – 8 байт
ar	Аналог відсутній	Еластичний масив – 42 байти без вмісту
ts	Аналог відсутній	Деревоподібна структура – 54 байти без вмісту
record	**record**	Запис – тип, що визначається користувачем. Розмір залежить від вмісту
enum	Перерахівний тип даних	Тип, що має декілька можливих значень. Для сумісності зі старим ПЗ. 4 байти
ref	Вказівник	Вказівник на область у пам'яті (змінну)
delegate	Вказівник на підпрограму	Підпрограма зворотного виклику
event	Аналог відсутній	Підпрограма зворотного виклику з підтримкою евристичних технологій

Примітка. В списку типів мови програмування МИША назви числових типів даних виділені жирним шрифтом, нечислових – виділені жирним шрифтом і підкреслені. В списку типів мови програмування Паскаль жирним шрифтом виділені ключові слова цієї мови програмування.

Засоби програмування алгоритмів

Мова МИША пропонує велику кількість зручних засобів з програмування алгоритмів. Так, до традиційної імперативної моделі (підпрограми та керівні конструкції мови Паскаль) додане управління пам'яттю з допомогою символьних імен (в МИШІ кожна змінна може мати символьне ім'я), евристичне та декларативне програмування.

МИША базована на традиційній "європейській октаві" алгоритмічної мови, проте кількість конструкцій суттєво розширена. Нові алгоритмічні засоби, що дають змогу у багатьох випадках обходитися

без використання рекурсії. Особливо помітна ефективність цих алгоритмічних засобів під час обробки деревоподібних структур як в плані швидкості розробки, так і в плані ефективності виконання. Так, в МИШІ можливий такий алгоритм:

```
ts<str> strts;
strts("Font")("Name") =
        "Times New Roman";
strts("Font")("Style") = "14";
through ts str(cnt, strts)
{
        Console.WriteLn(strts[cnt].Value);
}
```

Цей підхід спрощує написання алгоритмів обробки деревоподібних структур даних, а також забезпечує економне використання пам'яті комп'ютера за рахунок відмови від рекурсивної обробки (яка часто призводить до переповнення стеків). Водночас слід зазначити, що рекурсивні алгоритми на МИШІ, як і в декларативних мовах програмування, не допускають переповнення стеку, оскільки на віртуальний стек відводиться весь вільний простір ОЗП. Якщо в платформі .NET переважає явне розміщення даних у динамічній області пам'яті, то в МИШІ – автоматизоване виділення пам'яті в стеку підпрограми.

В МИШІ можливе наступне оголошення підпрограми (так виглядає на МИШІ алгоритм швидкого сортування Хоара):

```
generic(ptype T) ar <T> HS(ar<T> i):
Extract(min(0,1), i, i[0], 0)+i[0] + Extract(
max(0,1), i, i[0], 0);
```

Проте можливо і таке оголошення алгоритму швидкого сортування:

МИША дозволяє використовувати швидкісні динамічні евристики. Розглянемо наступний код:

```
/**MT::MAINFILE*/

using("mcl.hpp");

/*#
~goose~schm("Label!"); @12;
\Comment for user
$M
        schm("Hello from a heuristic!");
        ~12~@goose
$.
#*/

program
{
        self(); //Виконання коду евристики
        return EXIT_SUCCESS;
}
```

Виділена коментарями виду **/*#~~~#*/** евристика – це динамічний фрагмент програми. Набір технологій для роботи з засобами штучного інтелекту (зокрема, з нечіткою логікою) передбачає такі засоби:

- рефлексія МИША-програми – для перебудови ресурсоємних базових алгоритмів;
- рефлексія та динамічне генерування евристик – для побудови розумних систем на базі нейронних мереж та евристичних алгоритмів;
- синтаксичні конструкції та техно-логії опису знань та їх збереження;
- «штучнорозумовий» універсальний тип даних **ts**<**var**>.

```
ar<int> HoareSort(ar<int> i)
{
      ar<int> m,b; int h;
      h = i.PickFromConvaer();
      through(cnt, i) if(i[cnt]<=h) m+=i[cnt]; else b+=i[cnt];
      return HoareSort(m)+h+HoareSort(b);
}
```

У МИШІ важливою є проблема обробки динамічної пам'яті (“купи”). Як уже згадувалось, базовий підхід до створення програм у МИШІ передбачає передачу аргументів та збереження даних за значенням. Але можливі випадки, коли зручніше використати посилання (**ref**), особливо враховуючи звичку багатьох програмістів використовувати вказівники та подібні до них типи. МИША дає змогу написати такий код:

```
ref<TForm>r1=new TForm("Hello!",0,0,600,800,BorderStyle::Dialog);
TModalResult  r1sm = r1().ShowModal();
```

У МИШІ використана технологія інтелектуального керування динамічною областю пам'яті, яка базується на евристиках, та дає змогу гарантувати, що об'єкт буде знищений одразу після того, як він опиниться “поза зоною досяжності” (будуть втрачені всі посилання на нього). Використання технології підпрограм зворотного виклику та прямого доступу за вказівником у базовій інфраструктурі дає змогу уникнути зменшення швидкості програми та вчасно вивільняти критично важливі ресурси (для чого передбачено можливість оголошення підпрограм-деструкторів, які за призначенням аналогічні фіналізаторам у .NET/Java). Проте, як уже згадувалось, виклик деструктора в МИШІ чітко детермінований та виконується після останнього звертання до об'єкта. Тому в МИШІ немає необхідності у ручному звільненні ресурсів з допомогою спеціальних методів, як, наприклад, інтерфейс IDisposable у .NET.

Продуктивність генерованих програм

Транслятор МИШІ генерує ефективний асемблерний код. Використання технології платформної побудови дає змогу створювати незалежні від типу апаратного забезпечення програмні засоби, а також дослідити переваги та вади різних процесорних архітектур на різних типах задач.

Базове дослідження якості генерованого МИШОЮ коду складалося з трьох етапів – дослідження якості однопотокового коду, багатопотокового коду та аналіз конвеєризації програми для кластерної системи. Кожен дослід проводився п'ять разів, після чого в якості результату дослідження використовувалось середнє значення.

МИША містить якісну систему вимірювання продуктивності програми на основі відлагоджувача NEO. Вона дає змогу визначити швидкість програми в цілому та окремих її частин, а також показник використання ОЗП. Система вимірювання продуктивності інтегрована з транслятором, який використовує її дані у режимі самонавчання для визначення найкращої оптимізаційної стратегії. З метою оптимізації транслятор використовує три набори динамічних евристик для:

- визначення “вузьких місць” програми та порівняння продуктивності різних варіантів одного фрагмента коду на МИШІ (основні аналітичні евристики);
- генерування асемблерної програми у коді процесора МП-7Ю;
- конвертування програми з асемблера МП-7Ю в асемблер і386;
- зазначення паралельних фрагменттів програми для синхронізованого виконання на процесорі МП-7Ю-2;
- вибірка коротких послідовностей та сортування команд з метою оптимізації програми для суперскалярних архітектур (підтримуються моделі SIMD, CISC та EPIC, а також набори інструкцій MMX, SSE, SSE2, 3DNOW, AMD64, EM64T, PHC тощо);
- розподіл високорівневого про-

грамного коду на досить великі автономні блоки з метою паралельного їх виконання за макроконвеєрним принципом В.М. Глушкова;

- оптимізація використання пам'яті за рахунок неявного використання передачі вказівника на структуру даних замість власне структури;
- оптимізація використання пам'яті за рахунок побудови спеціалізованих евристик очистки динамічної пам'яті ("купи"). Ця методика дає змогу уникати традиційних вад технології "збирання сміття".

Розглянемо роботу інтелектуального оптимізатора МИШІ на прикладі алгоритму синтаксичного аналізу МИША-коду перед відправленням у нейронно-мережевий семантичний аналізатор. Обсяг алгоритму – 286 рядків коду, обсяг оброблюваного файлу – 15 КБ. При увімкнунему режимі роботи "без оптимізації" виконання програми займає час 1,23 сек. При увімкненні розпаралелення на комп'ютері з процесором AMD Athlon 64 X2 4800+ час виконання складає 0,96 сек.; наявне прискорення в 1,28 разів (це при тому, що алгоритм аналізу за деревоподібною структурою не може бути розділений на потоки за принципом макроконвеєрності). Рівномірність навантаження на ядра процесора – 2/0,91.

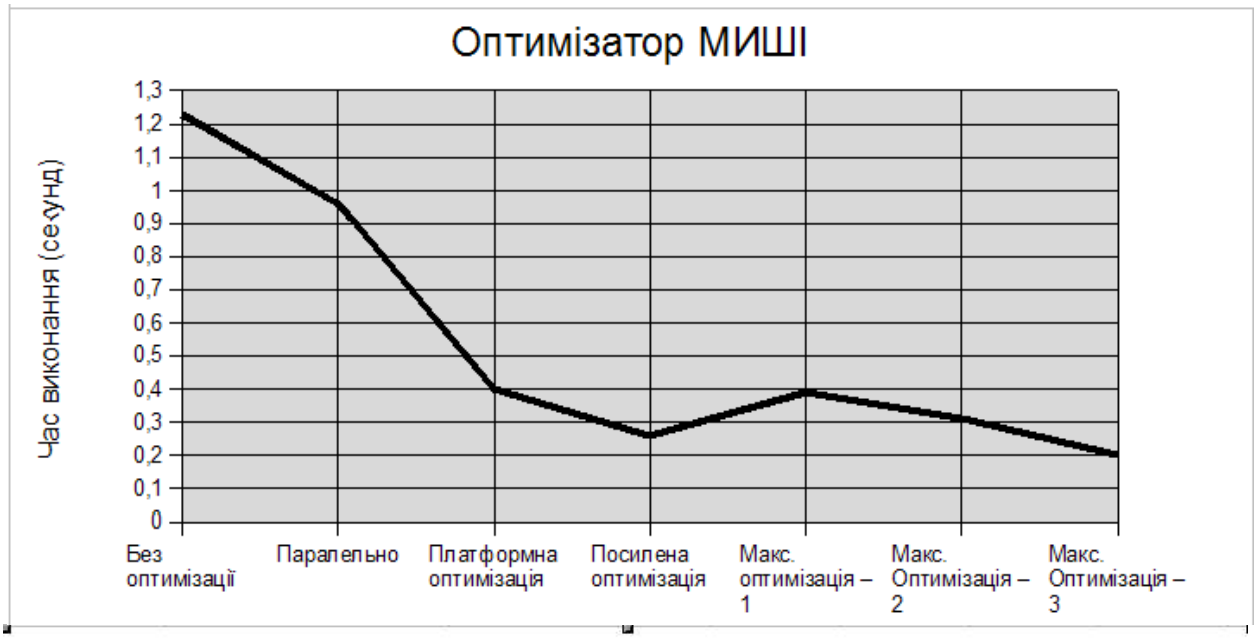
Оптимізатор МИШІ пропонує три стадії оптимізації:

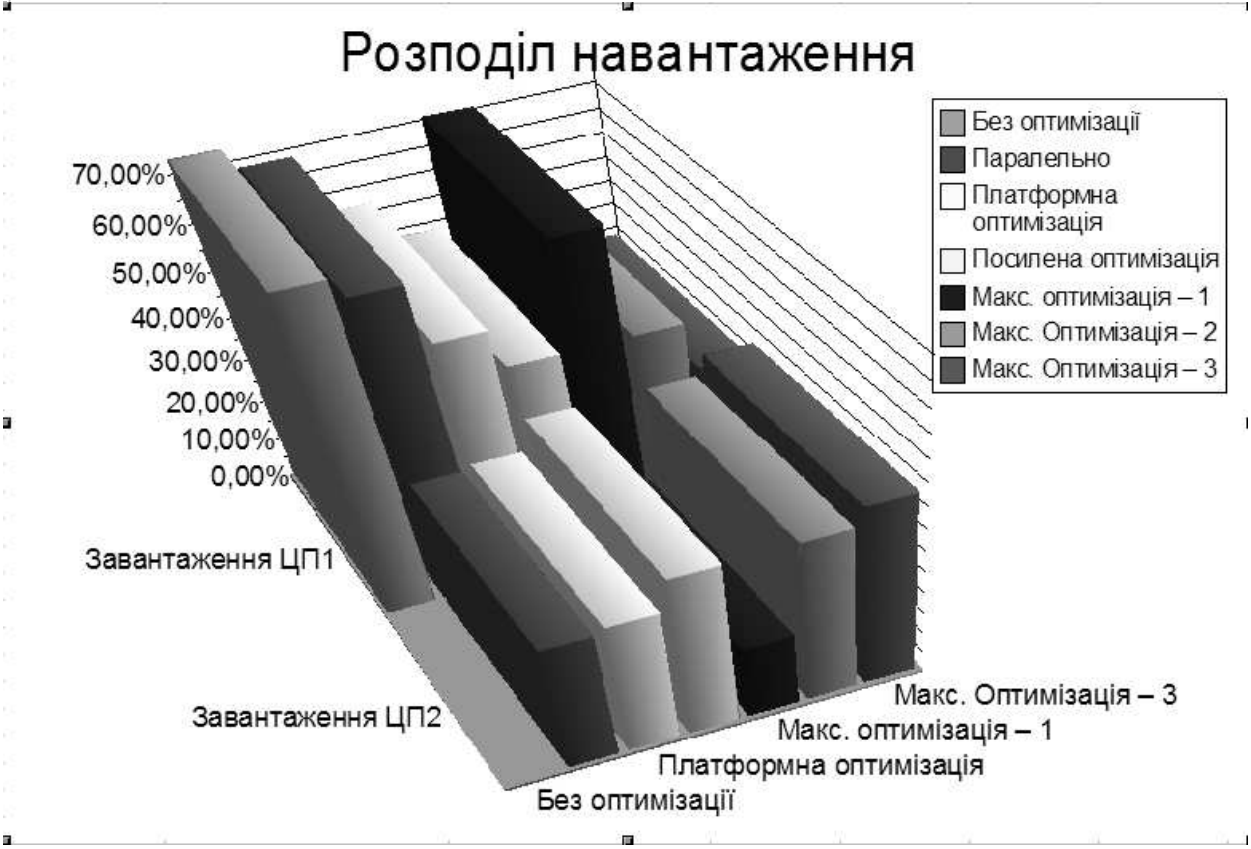
- оптимізацію на рівні асемблерного коду для певної платформи із технологією швидкого виконання;
- оптимізацію на рівні проміжного коду із зміною методів керування пам'яттю, вилученням фрагментів про-грами, що не впливають на результат її виконання та перевіркою умов циклів та інших параметрів алгоритмічних кон-струкцій (також включає в себе опти-мізацію на рівні асемблерного коду);
- повну оптимізацію, що передбачає виконання попередніх етапів оптимізації та велику кількість інтелектуальних аналітичних операцій, які забезпечують оптимальний баланс між продуктивністю та споживанням пам'яті, а також містить технології розподілу даних між потоками для більш якісного розпаралелення програмного коду і перевизначення вбудованих (inline) та генерування нових підпрограм.

Цим трьом стадіям оптимізації відповідають три пункти табл. 2 налаштуван-ня МИШІ – "Оптимізація", "Посилена оптимізація" та "Максимальна оптимізація".

Дослідимо ці три режими оптимізації. При використанні спеціалізованих для процесорних архітектур команд розпаралелювання і оптимізації коду на рівні асемблера швидкість виконання програми становить 0,4 сек – більш ніж в три рази швидше у порівнянні з режимом "без оптимізації". В цьому випадку рівномірність навантаження на ядра процесора становить 2/1,1, тобто розподіл виконано дещо краще, ніж при розпаралеленні коду без оптимізації. У режимі посиленої оптимізації помітне значне покращення показників продуктивності, а саме: загальний час виконання програми складає 0,26 сек.; рівномірність навантаження ядер процесора – 1,5/1. У режимі максимальної оптимізації продуктивність програми в 5,52 рази перевищує продуктивність програми без оптимізації. Як уже згадувалось, в цьому режимі транслятор "експериментує". Перше виконання програми дає результат 0,39 секунд із рівномірністю розподілу даних 2/0,5, друге – 0,31 сек. із рівномірністю навантаження 1,5/1, третє – 0,20 сек. із рівномірністю навантаження 0,9/1,19.

Найкраще отримані результати демонструють такі діаграми.





Спеціальний набір тестів наукових задач був використаний для визначення реальної продуктивності МИШІ у кластерній системі. Кластерна система мала таку конфігурацію:

- 9: Intel Celeron 3,06 Ghz (512 K L2 cache), 248 MB RAM.
- 2: Intel Celeron 1,7 Ghz (128 K L2 cache). 128 MB RAM.
- 1: Intel Core 2 Duo E6400, 2 GB RAM.
- 8: Intel Celeron 533 MHz (128 K L2 cache), 64 MB RAM.
- 1: AMD Athlon 64 X2 4800+, 8 GB RAM
- 1: Mobile AMD Turion 64 X2 MT-62, 2 GB RAM.

Загалом: 22 вузли кластера, 25 процесорів, 20 обчислювальних вузлів, 2 управляючі вузли.

Були отримані наступні результати:

	МИША М10 з динамічними евристичними технологіями	C++ стандартний дистрибутив Microsoft	.NET (C#) Бібліотека довгої арифметики для .NET (LAL) Стенфордського університету	J2EE (Java) Бібліотека довгої арифметики для J2EE (LAL) Стенфордського університету
Підрахунок простих чисел за методом решета Ератосфена (перші 10^{10} простих чисел)	500000 простих чисел за сек. в середньому Два потоки, автоматизоване розпаралелювання	Мова не дає можливості реалізувати даний алгоритм (недостатній обсяг типів даних).	273000 простих чисел за сек. в середньому Два потоки, ручне розпаралелювання	191000 простих чисел за сек. в середньому Два потоки, ручне розпаралелювання
Визначення об'єкта на растровому зображенні (алгоритм AI4-72/J)	12,3 сек. (автоматизоване розпаралелення)	14,1 сек. (MPI)	97 сек. (.NET remoting)	Не реалізовано через порушення в роботі JVM J2SE 6.0
BLES (система моделювання еволюції бактеріальних організмів), 2048 циклів виконання	5 год., 48 хромосом, 5 стартових клітин, 152 віртуальні хімічні елементи	Дослідження не проведене	3,5 год. 5 хромосом, 1 стартова клітина, 15 віртуальних хімічних елементів	Дослідження не проведене

Компонентно-орієнтована розробка програм

Максимально висока швидкість програмування з допомогою МИШІ досягається при одночасному застосуванні всіх засобів програмування. Розглянемо прийоми розробки з допомогою МИШІ на прикладі створення інтелектуального транслятора простої мови, подібної до мови Паскаль, для платформи .NET.

На вході транслятора – код програми, рядок символів, на виході – теж рядок, код мовою CIL, проміжні дані – це, власне, структура програми. Для реалізації синтаксичного аналізатора в МИШІ передбачено спеціальний компонент Tokenizer (за замовчуванням цей компонент налаштований на обробку файлів Паскаля).

Реалізація лексичного аналізатора як метасистеми одночасно демонструє і властивості мови МИША, і можливості широкого використання евристичного програмування і нейронних мереж. З метою забезпечення комплексного використання компонентів та швидкої паралельної розробки транслятор можна розподілити на такі частини:

а) лексичний аналізатор мови Паскаль;

б) синтаксичний аналізатор мови Паскаль;

в) ООП-ієрархія із представленням Паскаль-програми. (В МИШІ такі ієрархії мають назву репрезентативних, оскільки описують структуру оброблюваних даних);

г) ООП-ієрархія .NET-програми;

д) модуль трансляції (в) у (г);

е) модуль побудови CIL коду з (г).

Спроектована архітектура за суттю – комплекс для створення трансляторів для будь-якої мови програмування. При цьому

компоненти (г) і (е) є повторно застосовуваними. Тому для цих компонентів необхідно визначити метод взаємодії з усіма іншими.

Проектування ієрархії об'єктів виконується з допомогою спеціального засобу візуального програмування. Він дає змогу розробляти загальну структуру програмної системи, зокрема, і ієрархії об'єктів.

Найбільш складною проблемою є лексичний аналізатор. На вході отримуємо масив рядкових лексем, який необхідно перетворити на об'єктну структуру. Який метод роботи в цьому випадку оптимальний?

Детально проаналізувавши мову Паскаль, було визначено специфіку розташування лексем і ключових слів у цій мові. Так, лексемами верхнього рівня є **program, procedure, function, var, type**. Всі вони мають відповідники для завершення групи однієї лексеми (наприклад: **program** – команду **end**, **var** – оголошення типу, процедури чи функції, **type** – оголошення процедури, функції чи змінної, **procedure/function** – команда **end** підпрограми). В цьому випадку створюється деревоподібна стекова структура із набором змінних стану, підпрограм управління (реалізованих з допомогою евристик) та підпрограм-класифікаторів (реалізованих з допомогою евристик), що визначають виклик конкретної підпрограми управління.

Обсяг роботи не дає змоги детально описати прийоми програмування з допомогою МИШІ, зокрема, й подробиці реалізації транслятора. Зазначимо лише, що спроектований транслятор простої мови програмування займає 552 рядки коду (включно із універсальним генератором CIL та репрезентативною ієрархією .NET). Прикладні евристики – 120 рядків коду (з них 64 обслуговують універсальний кодогенератор).

Висновки

Розглянуто новий підхід до створення платформи програмування для наукових досліджень, що характеризується високою продуктивністю та захищеністю генерованого коду, а також засобами компонентно-орієнтованого та евристичного програмування. Продемонстровано техніку та переваги застосування такого підходу для програмування наукових задач.

1. *Java 2.* Patric Haughton, Herbert Shield – Sun Developers' Guide Press, San Diego, 2004.
2. *A. Troelsen,* The C# 2005 programming language and .NET 2.0. platform. Apress, 2006.
3. *Дорошенко А.Ю., Фінін Г.С., Цейтлін Г.О.* Алгеброалгоритмічні основи програмування. Об’єктна орієнтація і паралелізм. – К.: Наук. думка, 2004. – 458 с.
4. *Дорошенко А.Ю., Куйвашев Д.В.* Інтелектуалізація перенацілюваної оптимізуючої компіляції для мікропроцесорів цифрової обробки сигналів // Проблемы программирования. – 2002.– № 1/2. –С. 477 - 488.

Отримано 10.07.2007

Про авторів:

Дорошенко Анатолій Юхимович,
завідувач відділом, доктор фізико-математичних наук, професор,

Котюк Микола Васильович,

Ніколаєв Сергій Сергійович,
члени Київського територіального відділення Малої академії наук «Дослідник».

Місце роботи авторів:

Інститут програмних систем НАН України,
тел.: (38044)526 1538,
e-mail: dor@isofts.kiev.ua
Київ, вул. Січневого повстання 13.
e-mail: km@p5com.com