ISSN 2078–5097. Вісн. Львів. ун-ту. Сер. пр. матем. та інф. 2012. Вип. 18. С. 229–239
Visnyk of the Lviv University. Series Appl. Math. and Informatics. Issue 18. P. 229–239

УДК 531.8+62-50

МОДИФІКАЦІЯ ПАРАЛЕЛЬНОГО ГЕНЕТИЧНОГО АЛГОРИТМУ З ДІЙСНИМ КОДУВАННЯМ

Б. Литвин

Інститут прикладних проблем механіки і математики
ім. Я.С. Підстригача НАН України,
вул. Наукова, 3б, Львів, 79601, e-mail: dept25@iapmm.lviv.ua

Запропоновано модифікацію розпаралеленого генетичного алгоритму з дійсним кодуванням. Розроблена модифікація ґрунтується на процедурі розпаралелення типу master-slave, у якій розпаралелюється процес рекомбінації хромосом у наперед заданій кількості потоків. Побудований розпаралелений алгоритм орієнтовано на використання у сучасних багатоядерних комп'ютерних системах зі спільною пам'яттю. Досліджено прискорення (у часі) розв'язання задач у багатопотоковому середовищі Windows XP, проаналізовано вплив використаних структур даних на повноту завантаження CPU.

Ключові слова: генетичний алгоритм з дійсним кодуванням, паралельні обчислення.

1. ВСТУП

В основі генетичних алгоритмів (ГА) є стохастичний пошук, який використовує принципи природного відбору, репродукції та мутації [10, 18]. Ці алгоритми успішно використовують для розв'язання різноманітних наукових та інженерних задач і допомагають отримувати наближені розв'язки за прийнятний проміжок обчислювального часу. Проте зростання складності та розмірності прикладних задач суттєво збільшує час обчислення. Збільшення частоти сучасних процесорів не дає пропорційного приросту швидкодії [12]. Одним із ефективних підходів до збільшення швидкодії алгоритмів є їхнє розпаралелення [8]. Останнім часом у комп'ютерних системах широкого розповсюдження набули процесори з 4-6 ядрами, вже створюються експериментальні процесори, що мають до 80 ядер [5]. Все це сприяє зростанню зацікавленості до розробки розпаралелених алгоритмів для таких багатоядерних обчислювальних систем.

Для опису генетичних алгоритмів використовуємо запозичену з генетики термінологію [10]: фенотип, генотип, хромосома, пристосованість, особина, популяція, епоха, селекція (відбір), репродукція, схрещування та мутація. Під фенотипом розуміють компоненти вектора змінних, від яких залежить цільова функція. Алгоритм оперує з певним кодованим поданням аргументів цільової функції, яке називають генотипом; набір генів називають хромосомою. Пристосованість (або відповідність) – це оцінка (значення цільової функції), що визначається для кожної хромосоми. Під особиною розуміють хромосому, її пристосованість, а інколи й деякі інші параметри. Популяція – це набір особин на деякій епосі, під якою розуміємо конкретну ітерацію генетичного алгоритму. Селекція (або відбір) – це механізм вибору особин з найліпшою пристосованістю, які далі будуть брати участь у репродукції. Репродукція полягає в утворенні нових особин рекомбінацією генів батьківських хромосом, і відбувається за допомогою селекції та генетичних операцій схрещування, мутації. Схрещування дає змогу утворити декілька хромосом-нащадків комбінацією генетичного матеріалу двох або

більше батьків. Мутація змінює хромосому і забезпечує різноманітність генетичного матеріалу.

Типова схема генетичного алгоритму передбачає формування початкової популяції особин, з якої генерується нова популяція за допомогою застосування до особин генетичних операторів селекції, схрещування і мутації. Так триває доти, доки не виконається заданий критерій зупинки. Традиційно для ГА використовують бінарне кодування хромосом [10]. Однак багато практичних задач передбачають знаходження оптимальних параметрів, які набувають дійсні значення. Для розв'язання таких задач оптимізації широко популярними стали генетичні алгоритми з дійсним кодуванням (RCGA– Real-Coded Genetic Algorithm) [18, 19], в яких генотип збігається з фенотипом, що дає підстави уникнути операцій перекодування бітової стрічки у дійсні числа і навпаки. Для RCGA характерною особливістю є різноманіття операторів схрещування і мутації, поєднання яких допомагає підвищити ефективність розв'язання задач [11, 17].

Головна ідея більшості паралельних обчислень полягає в поділі задачі на частини й одночасному розв'язанні цих частин на кількох процесорах. Принцип "поділяй і владарюй" можемо застосовувати до ГА різними способами [3, 6–9, 13–16]. Деякі методи паралелізації використовуть одну популяцію, інші – ділять популяцію на частково ізольовані субпопуляції. Певні методи орієнтовані на масові комп'ютерні архітектури, інші – ліпше пристосовані до мікрокомп'ютерів з кількома потужними процесорними елементами. Існує три головні типи розпаралелювання ГА: "master-slave" з однією глобальною популяцією, дрібнозернисті з однією популяцією, крупнозернисті з багатьма популяціями [8].

У розпаралеленні за принципом "master-slave" використовують одну загальну популяцію, як і в послідовному ГА, лише обчислення пристосованості розподіляється між кількома процесорами. Такі типи ГА ще називаються глобальними паралельними, оскільки оператори селекції та схрещування використовують всю популяцію. Дрібнозернисті паралельні ГА пристосовані до масових паралельних комп'ютерних систем. Вони використовують одну просторово-структуровану популяцію. Селекції та схрещуванню піддаються особини з деякого околу (що визначається структурою популяції), але ці околи перетинаються. Це забезпечує деяку взаємодію між всіма особинами у популяції. Ідеальним випадком для цього типу розпаралелення є наявність одного процесорного елементи на кожну особину у популяції [8].

Багатопопуляційні ГА [9] більш ускладнені, оскільки містять декілька субпопуляцій, які час від часу обмінюються особинами. Цей обмін ще називають міграцією, він контролюється низкою параметрів. Багатопопуляційні ГА останнім часом дуже популярні, але цей клас алгоритмів до кінця ще не вивчений, тому що повністю не зрозумілий ефект міграції. Саме багатопопуляційність сьогодні є фундаментальною зміною, введеною у функціонування ГА. Такі алгоритми характеризуються іншою поведінкою, ніж класичні ГА, і відомі під різними іменами. Часто вони називаються "розподіленими" ГА, позаяк їх реалізують на MIMD комп'ютерах з розподіленою пам'яттю. Враховуючи, що співвідношення часу обчислень до часу комунікації досить велике, ці алгоритми часто називають крупнозернистими ГА. Зрештою, багатопопуляційні ГА відтворюють "острівну модель" у генетиці популяцій, яка розглядає частково ізольовані популяції. Отже, такі паралельні ГА часто називають "острівними".

Наведені вище методи розпаралелювання можуть поєднуватись. Зокрема, у багатопопуляційному ГА до субпопуляцій може застосовуватись паралелізація

"master-slave" або дрібнозернистого типу. Такий клас алгоритмів ще називають ієрархічними.

Зауважимо, що розпаралелення за принципом "master-slave" не змінює поведінку паралельного ГА порівняно з послідовним. Як і в послідовного ГА селекції та схрещуванню піддається вся популяція. Решта згаданих методів розпаралелення змінює поведінку алгоритму, бо у них схрещування обмежене до підмножини особин [6–9].

Ми подаємо розпаралелену модифікацію гібридного генетичного алгоритму з дійсним кодуванням [4]. Досліджується здатність до масштабування розпаралеленої версії RCGA – пропорційному збільшенні швидкодії зі збільшенням кількості процесорних елементів у обчислювальній системі.

2. РОЗПАРАЛЕЛЕННЯ ГІБРИДНОГО ГЕНЕТИЧНОГО АЛГОРИТМУ

Розглянемо задачу оптимізації

$$f(x, Param) \to \min;\ x \in \Omega \subset R^n,\ \Omega:\ a_i \le x_i \le b_i,\ i = 1,2,...,n, \qquad (1)$$

де *Param* – структура даних, яка містить константи та величини, які додатково визначають у процесі обчислення функції *f*. Вважаємо, що цільова функція

$$f(x, Param) \ge 0.$$

Це обмеження зумовлене особливістю реалізації операторів селекції [4]. Зазначимо, що у багатьох практичних задачах оптимізації, зокрема, задачах параметричної оптимізації режимів керування механічних систем [1, 2], цільова функція характеризує енерговитрати системи, тому таке обмеження прийнятне. Для тих самих задач *Param*, зокрема, містить лінійні та масоінерційні характеристики ланок, а також розгортку в часі вектора фазових координат і вектора керувань системи.

Наведемо структурну схему послідовного RCGA [4].

1. Приймемо номер епохи *Epoch*=0 і згенеруємо початкову популяцію $Popul_{Epoch}$ розмірністю N_{Popul}; обчислюємо пристосованість кожної особини.
2. За допомогою селекції утворюємо батьківський пул особин з хромосомами
$$C_j,\ j = 0,1,...,N_{Popul} - 1.$$
3. Зараховуємо ліпшу (з найменшим значенням функції пристосованості) особину з попередньої епохи в нову популяцію, застосовуючи генетичні оператори, формуємо решту особин нової популяції.
4. Перевіряємо критерій завершення: якщо він виконується, то переходимо до наступного кроку, інакше – переходимо до кроку 2.
5. Присвоюємо результуючому вектору оптимізації значення хромосоми найліпшої особини і завершуємо роботу алгоритму.

Розпаралелювати будемо 3-й крок RCGA – формування особин нової популяції, що виконуємо на підставі методики "master-slave" [8]. У розпаралелюваній модифікації RCGA послідовно використовується декілька відомих операторів схрещування і мутації, за якої заміна хромосом предків на хромосоми нащадків відбувалась лише у випадку поліпшення пристосованості особин у популяції. Результуючу популяцію розіб'ємо на частини і кожну частину будемо формувати у паралельних гілках. Отож, на відміну від класичної методики master-slave [8] розпаралелюється не лише обчислення пристосованості хромосом, а й їхнє формування. Нехай N_{CPU} – кількість паралельних гілок. Тоді у *j*-й гілці формуються особини з хромосомами

$$C_{i_j} \in \Omega,\ i_j = I_{j-1},...,I_j - 1,\ j = 1,...,N_{CPU},$$
$$I_0 = 1,\ I_k = I_{k-1} + N_{trd},\ k = 1,...,N_{CPU} - 1,\ I_{N_{CPU}} = N_{Popul} + 1, \quad (2)$$
$$N_{trd} = \left[N_{Popul} / N_{CPU}\right]$$

У співвідношеннях (2) квадратними дужками позначено цілу частину дійсного числа. Отже, розроблений алгоритм можна масштабувати на довільну кількість паралельних гілок залежно від кількості ядер CPU.

У розробленій модифікації ГА поєднано декілька відомих операторів схрещування і мутації: а) оператори схрещування – Unimodal Normal Distributed Crossover (UNDX), одноточкового, Simulated Binary Crossover, BLX-α, двоточкового, арифметичного; б) оператори мутації – неоднорідної, Гауса, по Mühlenbein, числового зсуву [11, 17, 18]. Ці оператори схрещування та мутації послідовно застосовують під час рекомбінації хромосом, за якої заміна хромосом предків на хромосоми нащадків відбувалась лише у випадку поліпшення пристосованості особин у популяції. Детальніше решту кроків RCGA описано в [4].

У багатопроцесорних системах зі спільною пам'яттю організація паралельного виконання команд в одному процесі, яким є прикладна програма, реалізується через механізм потоків (thread). Цей механізм підтримують сучасні системи програмування. Зокрема, у Delphi 7 потоки реалізуються у вигляді нащадка класу TThread, в якому у override-методі Execute реалізується підпрограма, що функціонує паралельно. Після завершення виконання потоку він знищується. Зауважимо, що кожен процес (прикладна програма) для ОС Windows XP містить принаймні 1 потік. Кванти часу система виділяє на активні потоки. Master-потік RCGA створює N_{CPU} slave-потоків, запускає їх на виконання і очікує їхнього завершення. Відповідні функції операційної системи (у нашому випадку WaitForMultipleObjects) переводять master-потік у неактивний стан до завершення всіх slave-потоків. Отже, під час очікування завершення slave-потоків система не виділяє квантів часу на master-потік. Під час створення *k*-го класу потоку йому, як параметри, передаються вказівники на масиви батьківського пулу хромосом і популяції наступної епохи, індекси $I_{k\text{-}1}$, I_k із співвідношень (2), а також копія структури *Param*. За такої будови RCGA slave-потоки не взаємодіють між собою, тому немає потреби використовувати іншу синхронізацію потоків.

Під час розв'язання практичних задач оптимізації, крім визначення мінімального значення функції мети і відповідних параметрів оптимізації, треба визначати низку інших величин, які можуть бути глобальними щодо функції мети. У задачі (1) ці величини позначено через *Param*. Для паралельного виконання кожен потік повинен використовувати окрему копію таких параметрів. Для розробленої модифікації RCGA функція мети має два параметри – вектор параметрів, що оптимізуються, і змінна без типу, яка є вказівником на довільну структуру даних, та містить згадані вище величини. У модуль RCGA передаються ці додаткові параметри, а також посилання на підпрограми створення і знищення динамічної копії додаткових параметрів цільової функції.

Зазначимо, що у випадку виведення інформації на екран через візуальні компоненти з функції мети, підпрограма RCGA повинна виконуватись також в окремому потоці. Інакше master-потік RCGA виконується у головному потоці VCL, і виникає ситуація, коли два потоки зупиняються і чекають завершення один одного (dead-lock).

3. ТЕСТУВАННЯ РОЗПАРАЛЕЛЕНОГО ГЕНЕТИЧНОГО АЛГОРИТМУ

Розроблений алгоритм реалізовано в інтегрованому середовищі Delphi 7. Тестування відбувалось на комп'ютерах з процесорами Intel Pentium dual-core T2080 1.73 ГГц (2 ядра) і Intel Core I5-750, 2.66 ГГц (4 ядра) в середовищі ОС Microsoft Windows XP SP3. Розроблений алгоритм тестували за кількості потоків від 1 до 5-ти, а також порівнювали з відповідною послідовною версією [4].

Запропоновану модифікацію ГА тестували для таких загальноприйнятих функцій [11]:

а) $$f_{Sph} = \sum_{i=1}^{n} (x_i - 1)^2$$

– узагальнена сфера;

б) $$f_{Ras} = an + \sum_{i=1}^{n} (x_i - 1)^2 - a\cos(\omega(x_i - 1))$$

– функція Растрігіна ($\omega = 2\pi,\ a = 10$);

в) $$f_{Ackl}(x) = -a \cdot \exp\left(-b \cdot \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - 1)^2} \right) - \exp\left(\frac{1}{n} \sum_{i=1}^{n} \cos \omega(x_i - 1) \right) + a + e$$

– функція Аклея ($a = 20$, $b = 0.2$, $\omega = 2\pi$);

г) $$f_{Roz} = \sum_{i=1}^{n-1} \left(100\left(x_{i+1} - x_i^2\right)^2 + \left(x_i - 1\right)^2 \right)$$

– узагальнена функція Розенброка.

Для тестових функцій (*а*), (*б*), (*г*) область визначення Ω в (1) задаємо такими значеннями $a_i = -6.12$, $b_i = 5.12$, для функції (*в*) – $a_i = -32.768$, $b_i = 32.768$, i=1,…,n. Для кожної з тестових функцій її мінімальне значення становить 0 і досягається у точці x_0=(1,…,1). Кількість змінних у тестових функціях приймали n=300. Зазначимо, що функції Растрігіна f_{Ras} і Аклея f_{Ackl} – є багатоекстремальні з одним глобальним мінімумом, а функція Розенброка є функцією типу яру [17].

Наша мета – дослідити прискорення при переході до багатопотокового виконання ГА. Основні параметри ГА приймали як у [4]: максимальна кількість епох 500, ймовірності схрещування $P_{Cross} = 0.95$, мутації $P_{Mut} = 0.85$, точність $\varepsilon = 0.01$. Числові експерименти проводили для розміру популяції $N_{Popul} = 200$ і $N_{Popul} = 2000$. Після кожної епохи виводили інформацію про найліпшу і середню пристосованість популяції. Після кожних 100 000 обчислень функції мети виводили її значення на екран. Деякі результати числових експериментів подано у табл. 1–3. У цих таблицях ГА з номером 1 відповідає послідовній модифікації алгоритму [4], 2 – описаній вище розпаралеленій модифікації, 3 – оптимізованій модифікації (про оптимізацію ГА далі). Час мінімізації тестових функцій послідовним ГА наведено у секундах і виділено жирним шрифтом. Для розпаралелених модифікацій наведено зменшення часу розв'язання задач порівняно з послідовною модифікацією алгоритму. Курсивом у табл. 2–3 виділені розв'язки, коли кількість потоків більша від кількості ядер процесора. Результати числових експериментів для комп'ютера з двоядерним процесором Intel Pentium dual-core T2080 подано у табл. 1.

Час обчислень для двох потоків зменшився на 43-47% порівняно з однопотоковим виконанням. Порівняно з послідовним ГА [4] час обчислень для розпаралеленої версії навіть при одному потоці зменшився на 3–10%. Для популяції $N_{Popul} = 2000$ час обчислень розпаралеленого ГА з одним потоком зростав на 9–17%.

Такий ефект простежувався і для комп'ютера з чотириядерним процесором Intel Core I5-750, для якого результати числових експериментів наведено у табл. 2. Для випадку 2-х потоків для чотириядерного процесора час обчислення також зменшився на 43-49%. У випадку 3-х потоків виграш в часі сягав 52-64%. Якщо для 4-х потоків для функцій f_{Ackl} і f_{Ras} час обчислень зменшився на 71%, то для f_{Sph} і f_{Roz} час обчислень порівняно з 3-ма потоками зріс, виграш становив лише 46–52% порівняно з послідовною модифікацією ГА [4]. Аналіз завантаження процесора (за допомогою диспетчера завдань Windows) виявив, що для 3-х потоків завантаження процесора було в межах 67%, а для 4-х потоків не перевищувало 80% для функції (*г*).

Зазначимо, що у [3] досліджували паралельний ГА для побудови ідентифікуючих послідовностей у цифрових схемах. Для обчислювальної двопроцесорної системи з 12 ядрами кратність приросту швидкодії була також менша за кількість ядер і для різних задач становила від 2,18 до 9 для 12 потоків.

Таблиця 1

Числове розв'язування тестових задач для комп'ютера з процесором Intel T2080, 2 ядра

ГА	N_{CPU}	N_{Popul}	f_{Sph}	f_{Ras}	f_{Ackl}	f_{Ros}
1	–	200	**33.718**	**86.969**	**44.204**	**66.250**
2	1		1.0662	1.0327	1.0265	1.1111
	2		2.0167	1.8142	1.8077	1.9503
3	1		1.3320	1.1597	1.1472	1.2318
	2		1.9834	2.2335	2.1128	2.1392
1	–	2000	**329.062**	**476.235**	**278.250**	**617.015**
2	1		0.8485	0.9159	0.9075	0.8792
	2		1.5999	1.6607	1.8118	1.8393
3	1		1.2140	1.1122	1.0936	1.1072
	2		1.9401	1.9430	2.0483	1.8428

Метою подальшої серії числових експериментів було з'ясування причин падіння приросту швидкодії зі збільшенням кількості потоків і неповного завантаження процесора для випадку, коли кількість потоків дорівнює кількості ядер процесора. Зауважимо, що функції (*б*) і (*в*) мають складніший аналітичний вигляд, містять трансцендентні функції, отже, час обчислень більший порівняно з функціями (*а*) і (*г*).

Для імітації складності практичних задач в підпрограми обчислення тестових функцій вводили цикли затримки (додаткові операції множення дійсних чисел). Результати цих числових експериментів для комп'ютера з процесором Intel Core I5-750 подано у табл. 3. Скорочення часу розв'язання задач порівняно з послідовною версією ГА при циклах затримки 1 000 і 10 000 додаткових операцій множення становило 3,17–3,58 та 3,72–3,95 разів. Неповне завантаження процесора свідчить про наявність ділянок коду, які у потоках виконуються послідовно. Аналіз вихідних текстів бібліотеки VCL і трасування програми в режимі команд процесора виявили, що код програми при багатопотоковому виконанні неявно використовує об'єкти синхронізації, зокрема критичні секції коду. При спробі захопити об'єкт синхронізації, захоплений іншим потоком, потік переходить у неактивний стан до звільнення цього об'єкта.

Таблиця 2

Числове розв'язування тестових задач для комп'ютера з процесором Intel Core I5-750, 4 ядра

ГА	N_{CPU}	N_{Popul}	f_{Sph}	f_{Ras}	f_{Ackl}	f_{Ros}
1	–		**16.718**	**46.282**	**23.609**	**32.093**
2	1		1.0963	1.0386	1.0456	1.0873
	2		1.8227	2.0484	2.0755	1.9359
	3		2.0937	2.9154	2.8890	2.5326
	4		1.8543	3.6569	3.5059	2.0788
	5	200	*1.8935*	*2.8291*	*3.0649*	*2.2545*
3	1		1.1554	1.0541	1.0716	1.1436
	2		1.9778	2.0714	2.0699	2.0917
	3		2.6813	2.9298	2.9115	3.0385
	4		3.1017	3.8418	3.6677	3.4695
	5		*2.6357*	*2.8869*	*2.8781*	*2.8767*
1	–		**162.563**	**253.015**	**148.235**	**305.625**
2	1		0.8758	0.9354	0.9407	0.8870
	2		1.6843	1.8012	1.9977	1.7317
	3		2.0194	2.5687	3.0175	2.1377
	4		1.7694	3.1461	3.5586	2.0082
	5	2000	*1.7916*	*2.8916*	*3.2268*	*1.9904*
3	1		1.0418	1.0043	1.0200	1.0399
	2		1.9024	1.8170	2.0345	1.9148
	3		2.5046	2.6858	2.8775	2.2707
	4		2.8387	3.5825	3.8301	3.0260
	5		*2.6501*	*2.7713*	*3.1414*	*2.7788*

Було розроблено програму для тестування поведінки деяких операцій при їхньому виконанні у багатопотоковому середовищі. Тестова програма відтворювала багатопотокову частину ГА, а саме наперед задану кількість ітерацій створення заданої кількості потоків, виконання в них деяких операцій, характерних для прикладних обчислювальних задач (доступу до елементів масивів, властивостей класів, створення і знищення динамічних змінних). Під час виконання тесту за допомогою диспетчера завдань Windows фіксували завантаження процесора. Тести проводили на комп'ютері з чотириядерним процесором Intel Core I5-750 для 4-х потоків.

У першій серії тестів у кожному потоці виконували задану кількість операцій створення і знищення динамічної змінної, класу, виконання присвоєння стрічкової константи. Для таких операцій завантаження процесора становило 25%, що свідчить про те, що ці операції виконуються послідовно в різних потоках. У наступній серії тестів масиви дійсних чисел заповнювали константами. У першому випадку тип масиву був фіксованої максимальної розмірності, динамічне виділення пам'яті під задану кількість елементів виконувалось функцією GetMem під час роботи потоку. Обчислювальні експерименти виявили, що завантаження процесора під час виконання 4-х потоків було в межах 97–99%, що свідчить про мінімальну синхронізацію потоків. У другому випадку використовували динамічні масиви, оголошені директивою `array of <тип елемента>`, виділення пам'яті

виконувалось функцією SetLength, де динамічний масив був: *а*) полем класу потоку і створювався під час виконання потоку; *б*) локальною змінною підпрограми потоку і створювався у потоці; *в*) полем класу потоку, створювався до запуску потоку; *г*) створювався до запуску потоку і у потік передавався вказівник на масив. Як з'ясувалось, у разі подання масивів (*а*)–(*в*) завантаження процесора було в межах 65–71%, 62–68% і 81–88%, відповідно, що свідчить про неявну синхронізацію між потоками. У випадку (*г*) завантаження процесора було в межах 99–100%. Трасування тестової програми в режимі команд процесора виявило існування у програмі інструкцій процесора з префіксом LOCK (блокування шини даних) і викликів функцій Windows для зайняття і звільнення об'єктів критичної секції.

Таблиця 3

Числове розв'язування тестових задач для комп'ютера з процесором Intel Core I5-750, 4 ядра з циклами затримки у тестових функціях

ГА	N_{CPU}	N_{Popul}	N_{Iter}	f_{Sph}	f_{Ras}	f_{Ackl}	f_{Roz}
1	–	200	10 000	**221.188**	**312.172**	**213.531**	**309.391**
2	1			1.0090	1.003	1.0104	1.0214
	2			1.9667	1.9995	2.0474	2.0166
	3			2.8662	2.9230	3.0905	2.9723
	4			3.7233	3.9958	4.1051	3.9578
	5			*3.0807*	*3.2167*	*3.4012*	*3.2455*
3	1			1.0138	1.0239	1.0046	1.0253
	2			1.9719	2.0165	2.0258	2.0267
	3			2.8466	3.1972	3.0953	3.0079
	4			3.7649	3.9221	4.0624	3.9777
	5			*3.2319*	*3.2460*	*3.3122*	*3.3057*
1	–	2 000	1 000	**419.828**	**528.593**	**381.235**	**578.482**
2	1			0.9344	0.9536	1.0426	0.9479
	2			1.8763	1.9000	1.8949	1.9335
	3			2.6912	2.6581	2.7077	2.7505
	4			3.1704	3.5882	3.5666	3.3188
	5			*3.0292*	*3.1455*	*3.1882*	*3.1401*
3	1			1.0225	1.0173	1.0109	1.0328
	2			1.9787	1.9237	1.9639	2.0053
	3			2.8627	2.8814	2.7317	2.9043
	4			3.6349	3.8535	3.5386	3.7157
	5			*2.9851*	*3.0012*	*2.9787*	*3.0631*

З урахуванням тестів було розроблено оптимізовану модифікацію розпаралеленого алгоритму, в якій всі тимчасові масиви, що використовували в операторах схрещування і мутації, створювали до запуску потоків (у пункті 1 алгоритму). У табл. 1–3 ця модифікація позначена як ГА 3. Як засвідчили числові експерименти, оптимізована модифікація ГА у випадку 4-х потоків завантажувала процесор на 97-99%, час розв'язання задач зменшився на 71–73% порівняно з послідовною версією.

Зауважимо, що при кількості потоків більшій, ніж кількість ядер процесора, час розв'язання задач збільшувався порівняно з випадком, коли кількість потоків

відповідала кількості ядер у системі. В оптимізованій версії розпаралеленого ГА час роботи в однопотоковому випадку був менший, ніж в послідовної версії ГА [4]. Зазначимо, що кількість обчислень відповідних функцій мети в однопотоковому виконанні у всіх наведених ГА і числовий розв'язок при однакових початкових значеннях генератора випадкових величин були ідентичними. Для кількості потоків, більших за 1, для отримання наближених розв'язків кількість обчислень функцій мети відрізнялась у межах від 2 до 10%.

4. ВИСНОВКИ

Отже, розроблено паралельну модифікацію гібридного генетичного алгоритму з дійсним кодуванням. За допомогою розпаралелення вдалось суттєво скоротити час розв'язання оптимізаційних задач. Числові експерименти виявили, що для ефективного використання можливостей розпаралелення у програмних модулях, які виконуються у контексті потоків, варто уникати операцій динамічного виділення пам'яті. У перспективі запропоновану паралельну числову схему ГА доцільно використати до розв'язання задач параметричної оптимізації руху нелінійних механічних систем, зокрема, локомоційних систем антропоморфного типу [1, 2].

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. *Бербюк В.Е.* Математическое моделирование и оптимизация ходьбы человека с протезированной голенью / В. Е. Бербюк, М. В. Демыдюк, Б. А. Литвин // Проблемы управления и информатики. – 2005. – № 3. – С. 128-144.
2. *Демидюк М.В.* Параметрична оптимізація ходи двоногого робота / М.В. Демидюк, Б.А. Литвин, Б.М. Голуб // Мат. методи та фіз. мех. поля. – 2005. – **48**, № 3. – С. 162–171.
3. *Иванов Д.Е.* Масштабируемый параллельный ГА построения идентифицирующих последовательностей для современных многоядерных ВС / Д.Е. Иванов // Управляющие системы и машины. – 2011. – № 1. – С. 25-32.
4. *Литвин Б.* Про одну модифікацію гібридного генетичного алгоритму з дійсним кодуванням / Б. Литвин // Вісник Львів. ун-ту. Серія. прикл. матем. та інформ. – 2009. – Вип. 15. – С. 313-324.
5. *Джиллеспи М.* Масштабирование программных архитектур для многоядерных вычислительных систем будущего / М. Джиллеспи // Intel® Software Network. – Режим доступу: http://software.intel.com/ru-ru/articles/scaling-software-architectures-for-the-future-of-multi-core-computing
6. *Alba E.* Heterogeneous Computing and Parallel Genetic Algorithms / Enrique Alba, Antonio J. Nebro, José M. Troya // Journal of Parallel and Distributed Computing. – 2002. – Vol. 62. – P. 1362-1385.
7. *Alba E.* Parallelism and Evolutionary Algorithms / Enrique Alba, Marco Tomassini // IEEE Transaction on Evolutionary Computation. –2002. – Vol. 6, No 5. – P. 443–462.
8. *Cantu-Paz E.* A Survey of Parallel Genetic Algorithms / E. Cantú-Paz // Calculateurs Paralleles, Reseaux et Systems Repartis. – 1998. – Vol. 10, No. 2. – P. 141-171.
9. *Eklund S.E.* A massively parallel architecture for distributed genetic algorithms / Sven E. Eklund // Parallel Computing. – 2004. – Vol. 30. – P. 647-676.
10. *Holland J.H.* Adaptation in Natural and Artificial Systems / J.H. Holland // The MIT Press: London, 1992. – 228 p.
11. *Herrera F.* Hybrid crossover operators for real-coded genetic algorithms: an experimental study / F. Herrera, M. Lozano, A. M. Sánchez // Soft Comput. – 2005. – No 9. – P. 280-298.

12. *Hikmat A*. The Impact of Overlocking the CPU to the Genetic Algorithm / Ahmed Hikmat, Azween Abdukkah // Int. J. of Computer Science and Network Security. – 2009. – Vol. 9, No. 5. – P. 175-180.
13. *Kobayashi K*. Adaptive Distributed Genetic Algorithms and Its VLSI Design / Kazutaka Kobayashi, Norihiko Yoshida, Shuji Narazaki // International Journal of Electrical and Computer Engineering. – 2009. – 4:5. – P. 322-325.
14. *Lim D*. Efficient Hierarchical Parallel Genetic Algorithms Using Grid Computing / Dudy Lim, Yew-Soon Ong, Yaochu Jin, Bernhard Sendhoff, Bu-Sung Lee // Future Generation Computer Systems.– Vol. 23, Issue 4. – 2007. – P. 658-670.
15. *Liu K*. Research and application of Distributed Parallel Genetic Algorithm Based on PC Cluster / Keyan Liu, Wanxing Sheng, Yunhua Li // IJCSNS International Journal of Computer Science and Network Security. – 2007. – Vol. 7, No 2. – P. 157-163.
16. *Rivera W*. Scalable Parallel Genetic Algorithms / Wilson Rivera // Artificial Intelligence Review. – 2001. – Vol. 16. – P. 153-168.
17. *Nakanishi H*. Searching perfomance of a real-coded genetic algorithm using biased probability distribution function and mutation / H. Nakanishi, H. Kinjo, N. Oshiro // Artif. Life Robotics. – 2007. – No 11. – P. 37-41.
18. *Michalewicz Z*. Genetic Algorithms + Data Structures = Evolution Programs. 3rd / Z. Michalewicz. – Springer-Verlag: New York, 1998. – 387 p.
19. *Wright A.H*. Genetic Algorithms for Real Parameter Optimization / A.H. Wright // Foundations of Genetic Algorithms / Rawlings G.J.E. ed., Morgan Kaufman. – San Mateo (California). – 1991. – Vol. 1. – P. 205-228.

Стаття: надійшла до редколегії 23.05.2011
доопрацьована 01.09.2011
прийнята до друку 15.09.2011

МОДИФИКАЦИЯ ПАРАЛЛЕЛЬНОГО ГЕНЕТИЧЕСКОГО АЛГОРИТМА С ВЕЩЕСТВЕННЫМ КОДИРОВАНИЕМ

Б. Литвин

Институт прикладных проблем механики и математики
им. Я.С. Подстригача НАН Украины,
ул. Научная, 3б, Львов, 79601, e-mail: dept25@iapmm.lviv.ua

Предложена модификация распараллеленного генетического алгоритма с вещественным кодированием. Разработанная модификация основана на процедуре распараллеливания типа master-slave, в которой происходит распараллеливание процесса рекомбинации хромосом в ранее заданном количестве потоков. Построенный генетический алгоритм ориентирован на использование в современных многоядерных компьютерах с общей памятью. Исследуется ускорение (во времени) решения задач в многопоточной среде Windows XP, анализируется влияние структур данных на полноту загрузки CPU.

Ключевые слова: генетический алгоритм с вещественным кодированием, параллельные вычисления.

MODIFICATION OF THE PARALLEL REAL-CODED GENETICS ALGORITHM

B. Lytwyn

Pidstryhach Institute for Applied Problems of Mechanics and Mathematics NAS Ukraine, Naukova str, 3-b, Lviv, 79601, e-mail: dept25@iapmm.lviv.ua

The parallelized modification of real-coded genetics algorithm is proposed. The proposed modification based on master-slave parallelizing procedure, in which chromosomes recombination process is parallelized in given threads number. The developed parallelized algorithm directed on using in modern multi-core computer systems with common memory. The acceleration (in time) of problems solving in multithread Windows XP environment is investigated. The influence of used data structures on utilization CPU is analyzed.

Key words: Real-Coded Genetic Algorithm, parallel computation.