УДК 681.3

СЕРЕДОВИЩЕ КОНСТРУЮВАННЯ АЛГОРИТМІЧНИХ ЗНАНЬ ТА ІНСТРУМЕНТАРІЙ СИНТЕЗУ ПРОГРАМ

О.А. Яценко

Інститут програмних систем НАН України,
03187, м. Київ, просп. Академіка Глушкова 40,
e-mail: aiyat@i.com.ua

Статтю присвячено огляду результатів, отриманих у рамках подальшого розвитку середовища конструювання алгоритмічних знань мультиобробки та інтегрованого інструментарію проектування і синтезу послідовних та паралельних об'єктно-орієнтованих програм. Інструментальні засоби грунтуються на апараті алгебри алгоритміки (стратегіях обробки, метаправилах конструювання схем). Розроблено низку паралельних алгоритмів сортування та пошуку, що входять до запропонованого середовища інструментарію для проектування та генерації програм символьної мультиобробки. Розглянуто перспективи застосування засобів Grid-обчислень для реалізації паралельних алгоритмів та програм.

The article is devoted to consideration of the results got within farther development of the environment of constructing of algorithmic knowledge of multiprocessing and integrated tool of designing and synthesis of consecutive and parallel object-oriented programs. The tools are based on algorithmics algebra apparatus (strategies of processing, metarules of schemes constructing). Several concurrent sort and search algorithms, which belong to the environment proposed, were developed. The tools are applied for designing and generation of programs of symbol multiprocessing. The prospects of applying of Grid-computation means for realization of parallel algorithms and programs are considered.

Вступ

В основу пропонованих у даній статті алгебраїчних засобів проектування та синтезу алгоритмів і програм покладено концепцію алгебри алгоритміки (АА) [1]. Дана алгебра бере свій початок від концепції систем алгоритмічних алгебр (САА) Глушкова [2] і є перспективним напрямком, що розвивається у рамках української алгебро-кібернетичної школи. АА займає власну нішу в алгебраїчній алгоритміці, яка останнім часом отримала поширення на Заході. Одним із напрямків досліджень алгоритміки є розробка засобів подання, накопичення, конструювання і класифікації алгоритмічних знань, що відносяться до задач символьної обробки (сортування, пошук, мовне процесування). Подання таких знань асоційоване з описом алгоритмів за допомогою схем та використанням метаправил конструювання, орієнтованих на породження нових алгоритмічних знань. У даній статті розглянуті результати, отримані у рамках подальшого розвитку засобів проектування і генерації алгоритмів і програм, а саме середовища конструювання алгоритмічних знань (СКЗ) символьної мультиобробки. Згадане середовище призначене для підтримки інтегрованих алгебро-алгоритмічних моделей, до яких, зокрема, відносяться розподілені гіперсхеми – високорівневі подання у алгебрі алгоритмів, що призначені для генерації послідовних та паралельних регулярних схем (РС) [3]. У рамках розробки СКЗ розпаралелено низку паралельних алгоритмів символьної обробки, що входять до його складу. СКЗ реалізоване у розробленому інтегрованому інструментарії проектування та синтезу об'єктно-орієнтованих програм. Інструментальні засоби застосовано для розробки алгоритмів і програм символьної мультиобробки (паралельний пошук). Розглянуто також можливості застосування інструментарію Grid-обчислень (Globus Toolkit [4]) для реалізації паралельних програм у сучасних мультипроцесорних комплексах.

1. Проектування паралельних алгоритмів у середовищі конструювання алгоритмічних знань символьної мультиобробки

Розроблене середовище конструювання знань орієнтоване на проектування схем алгоритмів символьної мультиобробки, а також синтез програм обраною цільовою мовою [3]. Для подання паралельних алгоритмів застосовуються операції модифікованих САА (САА-М) [2]. У СКЗ спільно використовуються три форми подання алгоритмів при їх проектуванні: аналітична (формули в алгебрі алгоритмів), природно-лінгвістична (САА-схеми) і графова (граф-схеми Калужніна). Середовище складається із таких компонентів: стратегій обробки, що описують класи алгоритмів; метаправил конструювання схем (згортка, розгортка, трансформація та ін.), які забезпечують породження нових алгоритмічних знань; базисних предикатів та операторів, поданих у згаданих трьох формах, а також їх програмних реалізацій; схем алгоритмів символьної обробки; розподілених гіперсхем.

У даному розділі побудовано низку послідовних та асинхронних алгоритмів сортування масивів і пошуку у файлах, що входять до складу розробленого СКЗ. Асинхронні схеми отримані у результаті розпаралелення послідовних алгоритмів, наведених у [1]. Під терміном "розпаралелення" мається на увазі перетворення послідовних алгоритмів у паралельні на основі застосування метаправил конструювання схем, зокрема трансформації.

Розглянемо паралельний алгоритм двостороннього асинхронного бульбашкового сортування *БСОРТ/П*, що входить до складу СКЗ. Розмітка масиву, який підлягає сортуванню, має вигляд M: $H\ У_1\ a_1\ a_2\ ...\ a_n\ У_2\ K$, де H і K – маркери, що відзначають початок і, відповідно, кінець масиву M; $У_1$ і $У_2$ – вказівники, що переміщаються в процесі обробки назустріч один одному. Паралельна регулярна схема (ПРС) даного алгоритму є такою:

$$БСОРТ/П = СТАРТ * УСТ(У_1, H) * УСТ(У_2, K) * \{[УМ]\ ІНІЦ\_КТ * (П> \dot\vee\ П<)\} * ФІН;$$

$$П> = \{[d(У_1, K)]\ ТРАНСП\_Л\} * УСТ(У_1, H) * Т(ОБР\_ЗАК(1)) * S(ОБР\_ЗАК(2));$$

$$П< = \{[d(У_2, H)]\ ТРАНСП\_П * ([d(У_1, У_2) = 1]\ S(d(У_1, У_2) = 0), E)\} * Т(ОБР\_ЗАК(2)) * S(ОБР\_ЗАК(1));$$

$$ТРАНСП\_Л = ([l > r|\ У_1]\ ТРАНСП(l, r|\ У_1), E) * P(У_1);$$

$$ТРАНСП\_П = ([l > r|\ У_2]\ ТРАНСП(l, r|\ У_2), E) * L(У_2),$$

де *СТАРТ* – оператор ініціалізації; $УСТ(У_1, H)$ – установка вказівника $У_1$ на маркер H; *УМ* – умова істинна, якщо масив M відсортований; *ІНІЦ_КТ* – ініціалізація контрольних точок; *ФІН* – заключний оператор; $d(У_1, K)$ – умова істинна, якщо вказівник $У_1$ досяг маркера K; $Т(ОБР\_ЗАК(1))$ – контрольна точка для фіксації моменту завершення обробки в першій гілці *П>*; $S(ОБР\_ЗАК(2))$ – синхронізатор, що здійснює затримку обчислень доти, поки не буде виконана умова $ОБР\_ЗАК(2)$ завершення обробки у другій гілці *П<*; $d(У_1, У_2) = 1$ – умова істинна, якщо відстань між вказівниками $У_1$ і $У_2$ (кількість елементів масиву між ними) дорівнює 1; $l > r|\ У_1$ – умова істинна, якщо відношення > виконується для елементів, які знаходяться зліва і справа від вказівника $У_1$; $ТРАНСП(l, r|\ У_1)$ – оператор транспозиції елементів, сусідніх зі вказівником $У_1$; E – тотожний оператор; $P(У_1)$, $L(У_2)$ – оператори зсуву вказівників $У_1$ і $У_2$ вправо і вліво на один елемент по масиву M відповідно.

Сутність ПРС *БСОРТ/П* полягає у спільному функціонуванні зустрічних гілок *П>* і *П<* бульбашкового сортування, що обробляють масив M з протилежних кінців. За допомогою складених операторів *ТРАНСП_Л* і *ТРАНСП_П* у гілках реалізується умовна транспозиція сусідніх елементів масиву та зсув вказівників $У_1$ та $У_2$ вправо і вліво відповідно. При досягненні вказівниками критичного стану (умова $d(У_2, У_1) = 1$ істинна) гілка *П<* переходить у стан очікування до моменту сполучення вказівників (умова $d(У_2, У_1) = 0$ істинна), у той час як *П>* продовжує функціонувати. Далі обидві гілки досягають протилежних кінців масиву. Описаний цикл повторюється знову аж до виконання умови упорядкованості масиву, і після досягнення вказівниками $У_1$ і $У_2$ протилежних кінців масиву алгоритм *БСОРТ/П* завершує роботу.

Для паралельного алгоритму бульбашкового сортування *БСОРТ/П* максимальна кількість операцій порівняння елементів масиву є в 1.6 разів меншою, ніж для відповідного послідовного алгоритму, а максимальна кількість переміщень – в 2 рази меншою, ніж для послідовного.

У результаті трансформації бульбашкового сортування може бути отримано алгоритм човникової обробки масиву, кращий за часом виконання. У асинхронному алгоритмі човникового сортування *ЧОВНСОРТ/П*, схему якого наведено нижче, по вхідному масиву формуються два упорядкованих підмасиви за допомогою складеного оператора *ЧОВНИК><*, після чого здійснюється злиття згаданих підмасивів за допомогою схеми *ЗЛИТТЯ><*. Розмітка оброблюваного масиву M є такою: M: $H\ У_1\ У_2\ a_1\ a_2\ ...\ a_n\ У_3\ У_4\ K$.

ПРС алгоритму *ЧОВНСОРТ/П* має вигляд

$$ЧОВНСОРТ/П = СТАРТ * УСТ(У_1, H) * УСТ(У_2, H) * УСТ(У_3, K) * УСТ(У_4, K) * ЧОВНИК>< * * ЗЛИТТЯ>< * ФІН,$$

$$ЧОВНИК>< = \{[d(У_1, У_3) = 0]\ ЧОВН>\} * Т(d(У_1, У_3) = 0)\ \dot\vee\ \{[d(У_1, У_3) \le k]\ ЧОВН<\} * S(d(У_1, У_3) = 0),$$

$$ЧОВН> = \{[(l > r|\ У_1) \vee (d(У_1, У_3) = 0)]\ P(У_1) * P(У_2)\} * \{[(l < r|\ У_2) \vee (d(У_1, У_3) = 0)]\ ТРАНСП(l, r|\ У_2) * * L(У_2)\} * УСТ\ (У_2, У_1),$$

$$ЧОВН< = \{[(l > r|\ У_3) \vee (d(У_1, У_3) = 1)]\ L(У_3) * L(У_4)\} * \{[(l < r|\ У_4) \vee (d(У_1, У_3) = 1)]\ ТРАНСП(l, r|\ У_4) * * P(У_4)\} * УСТ\ (У_4, У_3),$$

$$ЗЛИТТЯ>< = P(У_3) * P(У_4) * \{[(l \le r|\ У_1) \wedge (l \le r|\ У_3)]\ ІНІЦ\_КТ * (ВСТАВ> \dot\vee\ ВСТАВ<)\},$$

$$ВСТАВ> = \{[l \le r|\ У_2]\ ТРАНСП(l, r|\ У_2) * L(У_2)\} * УСТ(У_2, У_1) * Т(ОБР\_ЗАК(1)) * S(ОБР\_ЗАК(2)),$$

$$ВСТАВ< = S((l \le r|\ У_4) \wedge (d(У_4, У_1) > 1) \vee ОБР\_ЗАК(1)) * \{[l \le r|\ У_4]\ ТРАНСП(l, r|\ У_4) * P(У_4)\} * * УСТ(У_4, У_3) * Т(ОБР\_ЗАК(2)) * S(ОБР\_ЗАК(1)).$$

Схему складеного оператора ЧОВНИК>< отримано у результаті розпаралелення послідовного маятникового човникового сортування, наведеного в [1]. В основу схеми покладено складені оператори ЧОВН> і ЧОВН< лівобічного і правобічного човникового сортування масиву. ЧОВН> здійснює пошук зліва направо невпорядкованого елемента l > r по вказівнику У1 і установку його на місце зліва від У1 згідно човникової стратегії обробки. При цьому ЧОВН< здійснює пошук справа наліво невпорядкованого елемента l > r по вказівнику У3 і встановлює його на місце праворуч від У3. При зближенні вказівників У1 і У3 на відстань, що не перевищує коефіцієнта синхронізації k = 1, другий процес переходить у стан очікування, що продовжується доти, поки лівобічний процес не завершить обробку. Функціонування лівобічного процесу завершується в

момент злиття вказівників $У_1$ і $У_3$, що фіксують ліворуч від $У_1$ і праворуч від $У_3$ відсортовані кожним процесом підмасиви. Тіло зовнішнього циклу схеми *ЗЛИТТЯ*>< складається з двох зустрічних асинхронних гілок *ВСТАВ*> і *ВСТАВ*< для вставки неупорядкованих елементів на місце, установки вказівників $У_2$ і $У_4$ на $У_1$ і $У_3$, відповідно, а також синхронізації зазначених паралельних процесів. Функціонування схеми *ЗЛИТТЯ*>< завершується при виконанні умови $l \le r|\ У_1$ так, що після зняття вказівників і маркерів отримуємо відсортований масив.

Максимальна кількість порівнянь і переміщень в отриманому паралельному алгоритмі *ЧОВНСОРТ/П* в 1.33 рази менша за кількість згаданих операцій у відповідному послідовному алгоритмі.

Більш ефективним за часом виконання є алгоритм асинхронного сортування Шелла *ШЕЛЛСОРТ*(k)-*А*. Сутність даного алгоритму полягає в поділі масиву, що підлягає сортуванню, на групи елементів, які не перетинаються і віддалені один від одного на відстань k, та сортуванні цих груп. Далі параметру k присвоюється значення цілої частини від ділення k на 2 з новим поділом масиву на групи і т. д., аж до упорядкування масиву. Розмітка оброблюваного масиву M є такою: M: $Н\ У_1\ a_1\ a_2\ ...\ a_n\ У_2\ К$.

ПРС асинхронного сортування Шелла має вигляд

$$ШЕЛЛСОРТ(k)\text{-}А = СТАРТ * (k := n) * \{[УМ]\ (k := [k/2]) * ІНІЦ\_КТ * \dot{\bigvee_{i=1}^{k}} (ЧОВНИК_i(k) * * Т(ОБР\_ЗАК(i))) * S(ОБР\_ЗАК)\} * ФІН;$$

$$ЧОВНИК_i(k) = УСТ\_Л(У_{3i-2}, i) * УСТ\_Л(У_{3i-1}, i) * УСТ\_Л(У_{3i}, i) * \{[d(У_{3i-2}, У_{3i-1}) = k]\ Р(У_{3i-1}) * Р(У_{3i})\} * * ([r|\ У_{3i-2} > r|\ У_{3i-1}]\ ТРАНСП(r|\ У_{3i-2}, r|\ У_{3i-1}), Е) * \{[d(У_{3i}, К)]\ Р(У_{3i-2}, k) * Р(У_{3i-1}, k) * Р(У_{3i}, k) * * \{[r|\ У_{3i-2} \le r|\ У_{3i-1}]\ ТРАНСП(r|\ У_{3i-2}, r|\ У_{3i-1}) * L(У_{3i-2}, k) * L(У_{3i-1}, k)\} * \{[d(У_{3i-1}, У_{3i}) = 0]\ Р(У_{3i-2}, k) * * Р(У_{3i-1}, k)\}\};$$

де $УСТ\_Л(У_{3i-2}, i)$ – установка вказівника $У_{3i-2}$ зліва від i-го елемента масиву M; $d(У_{3i-2}, У_{3i-1}) = k$ – умова істинна, якщо відстань між вказівниками $У_{3i-2}$ і $У_{3i-1}$ дорівнює k; $ТРАНСП(r|\ У_{3i-2}, r|\ У_{3i-1})$ – перестановка місцями елементів, що стоять справа від вказівників $У_{3i-2}$ і $У_{3i-1}$; $ЧОВНИК_i(k)$ – алгоритм човникового сортування підмасиву елементів, що стоять на i-й, $i + k$-й, $i + 2k$-й, … позиціях в оброблюваному масиві M, $i = 1, …, k$; $S(ОБР\_ЗАК)$ – синхронізатор, що реалізує очікування моменту завершення обчислень у всіх k паралельних гілках. Алгоритм $ЧОВНИК_i(k)$ сортує підмасив $M_i(k)$ так, що в початковому стані зліва від його першого елемента встановлені вказівники $У_{3i-2}$, $У_{3i-1}$, $У_{3i}$. ПРС *ШЕЛЛСОРТ*(k)-*А* представляє алгоритми багатошарового сортування зі спадним кроком (параметр k) та зчепленими підмасивами, що асинхронно сортуються у гілках $ЧОВНИК_i(k)$. Замість алгоритму $ЧОВНИК_i(k)$ можуть бути використані відповідні модифікації інших послідовних або паралельних алгоритмів сортування.

Відзначимо, що за алгоритмами сортування можуть бути отримані відповідні алгоритми пошуку, і навпаки, із використанням метаправила переорієнтації.

Розглянемо алгоритм *ЧОВНП/П* паралельного човникового пошуку записів у файлі за масивом запитів, який може бути отриманий із схеми човникового сортування. Пошук виконується у відсортованому файлі F_s, розмітка якого має вигляд F_s: $Н\ У_i\ z_1\ z_2 \dots z_n\ К$, де $Н$ і $К$ – маркери, що позначають відповідно початок і кінець файлу F_s, z_i – запис у файлі, $У_i$ – вказівник. Пошук записів у файлі F_s здійснюється за масивом запитів Q: $Н'\ q_1\ q_2\ ...\ q_m\ К'$, де кожний запит q_i: $D \to \{0, 1\}$ $(i = 1, 2, …, m)$ є предикатом, визначеним на множині D елементів файлу F_s; $Н'$ і $К'$ – маркери. Якщо $q_i(z) = 1$ для деякого елемента $z \in D$, то вважається, що елемент z задовольняє запиту q_i.

Схему *ЧОВНП/П* отримано у результаті розпаралелення РС послідовного човникового пошуку *ЧП*, наведеної в [1], аналогічно до паралельного алгоритму бульбашкового пошуку [3], а саме було розпаралелено зовнішній цикл згаданої РС, у якому здійснювався послідовний перехід від обробки одного запиту із масиву Q до іншого. Розроблена ПРС алгоритму складається з m паралельних гілок $П_i$, кожна з яких реалізує пошук записів, які задовольняють i-му запиту

$$ЧОВНП/П = \dot{\bigvee_{i=1}^{m}} П_i,$$

$$П_i = УСТ(У_i, Н) * \{[r|\ У_i = r_x]\ Р(У_i)\} * ([\gamma]\ А * SUMM, NOT) * Т(u_i) * ([i = m]\ S(u_1 \wedge u_2 \wedge ... \wedge u_{m-1}) * В, Е),$$

де $r|\ У_i = r_x$ – умова істинна, якщо запис $r|\ У_i$, який оглядається вказівником $У_i$ (тобто знаходиться справа від $У_i$), співпадає з r_x – записом, що знаходиться у передбачуваному місці шуканого запису у файлі F_s; γ – умова істинна, якщо елемент $r|\ У_i$ файлу F_s, що оглядається вказівником $У_i$, задовольняє i-му запиту; $А$ – оператор обробки знайденого запису; $SUMM$ – оператор підсумовування результатів пошуку; NOT – оператор реакції на відсутність у файлі F_s шуканого запису; $Т(u_i)$ – контрольна точка за умовою u_i завершення виконання запиту у i-й гілці; $S(u_1 \wedge u_2 \wedge ... \wedge u_{m-1})$ – синхронізатор, який виконується в m-й гілці і реалізує очікування моменту завершення обчислень у гілках $i = 1, 2, ..., m-1$; $В$ – оператор виводу результатів пошуку.

В алгоритмі *ЧОВНП/П* здійснюється сканування вказівника $У_i$ по файлу F_s зліва направо із перевіркою умови $r|\ У_i = r_x$. При цьому координата x запису r_x визначається таким чином. Позначимо відповідно a_l та a_r елементи файлу F_s, які знаходяться зліва та справа від поточного елемента $r|\ У_i$, який оглядається вказівником $У_i$. Якщо задане у оброблюваному запиті слово q_i задовольняє відношенням $z_l < q_i < z_r$, то елемент $r|\ У_i$ знаходиться у місці x файлу і умова $r|\ У_i = r_x$ істинна. У випадку, коли вказівник $У_i$ оглядає перший елемент

файлу, для істинності зазначеної умови запис z_r повинен задовольняти відношенню $q_i < z_r$. Якщо $У_i$ оглядає останній елемент файлу, то елемент z_l повинен задовольняти відношенню $z_l < q_i$. Таким чином в даному алгоритмі при переміщенні вказівника $У_i$ по F_s від маркера H в напрямку зліва направо здійснюється пошук місця розташування у файлі шуканого запису. Потім, якщо умова γ істинна (шуканий запис знайдено), здійснюється його обробка (виконання оператора A). У протилежному випадку виконується оператор *NOT*, який сигналізує про відсутність шуканого слова в F_s.

ПРС *ЧОВНП/П* може бути трансформована в ПРС *АЛЬТП/П* альтернативного пошуку елементів у файлі F_s, що задовольняють запитам із масиву Q:

$$АЛЬТП/П = \dot{\bigvee_{i=1}^{m}} П_i,$$

$$П_i = УСТ\_СЕРЕД(У_i, H, K) * ([r|\ У_i > x]\ \{[r|\ У_i = x]\ L(У_i)\}, \{[r|\ У_i = x]\ P(У_i)\}) * ([\gamma]\ A * SUMM, NOT) * * T(u_i) * ([i = m]\ S(u_1 \wedge u_2 \wedge ... \wedge u_{m-1}) * B, E),$$

де *УСТ_СЕРЕД*($У_i$, H, K) – установка вказівника $У_i$ усередині між маркерами H і K. В наведеному алгоритмі вказівник $У_i$ переміщується від середини файлу F_s зліва направо або справа наліво залежно від координати x розташування у файлі шуканого слова.

Переінтерпретація ПРС *АЛЬТП/П* приводить до ПРС *БІНП/П* бінарного пошуку:

$$БІНП/П = \dot{\bigvee_{i=1}^{m}} П_i,$$

$$П_i = УСТ(У_i, Н) * УСТ(У_{i1}, Н) * УСТ(У_{i2}, К) * \{[\gamma \vee ЗЛ(У_{i2}, У_{i1}) \vee d(У_{i2}, К)]\ УСТ\_СЕРЕД(У_i, У_{i1}, У_{i2}) * * ([r|\ У_i > q_i]\ УСТ(У_{i2}, У_i) * L(У_{i2}), ([r|\ У_i < q_i]\ УСТ(У_{i1}, У_i) * Р(У_{i1}), Е))\} * ([\gamma]\ А * SUMM, NOT) * Т(u_i) * * ([i = m]\ S(u_1 \wedge u_2 \wedge ... \wedge u_{m-1}) * В, Е),$$

де *ЗЛ*($У_{i2}$, $У_{i1}$) – умова істинна, якщо вказівник $У_{i2}$ знаходиться лівіше вказівника $У_{i1}$ у файлі F_s; *УСТ_СЕРЕД*($У_i$, $У_{i1}$, $У_{i2}$) – установка вказівника $У_i$ усередині між $У_{i1}$ і $У_{i2}$; q_i – запит, якому повинне задовольняти шукане у файлі F_s слово. Алгоритм бінарного пошуку базується на фіксації медіани (середини) між вказівником $У_i$ та найближчим зліва (справа) вказівником $У_{i1}$ ($У_{i2}$) з наступною установкою вказівника $У_{i1}$ ($У_{i2}$) в точці медіани. Спочатку запит q_i порівнюється із записом ($r|\ У_i$), що міститься у середині файлу. Результат порівняння або приводить до розв'язання задачі, або дозволяє визначити, у якій частині файлу продовжувати пошук – зліва або справа від вказівника $У_i$. Описаний процес застосовується до отриманого "вкороченого" наполовину файлу і т. д.

Розроблені алгоритми паралельного пошуку потребують у найкращому випадку приблизно в m разів менше операцій порівняння за послідовні, де m – кількість паралельно оброблюваних запитів.

2. Інтегрований інструментарій проектування і синтезу програм та його застосування

Розглянуте у попередньому розділі середовище конструювання алгоритмічних знань реалізоване у інтегрованому інструментарії проектування та синтезу класів послідовних і паралельних програм [3]. Інструментарій забезпечує побудову САА-схем на основі методу діалогового конструювання синтаксично правильних програм (ДСП-методу) та інтегрує аналітичну, природно-лінгвістичну і графову форми подання алгоритмів при їх проектуванні. За побудованими схемами алгоритмів здійснюється синтез програм у сучасних об'єктно-орієнтованих середовищах програмування (Java, C++ та ін.). У процесі проектування та генерації програм спільно зі створеним інструментарієм застосовуються засоби візуалізованого проектування – мова UML і система Rational Rose [5]. Інтегрований інструментарій складається із компонентів: діалогового конструктора синтаксично правильних схем алгоритмів і синтезу програм (ДСП-конструктора), редактора граф-схем, трансформатора алгоритмів, генератора САА-схем, середовища конструювання алгоритмічних знань. Більш детально компоненти інтегрованого інструментарію розглянуто у [3]. У підрозділах 2.1 і 2.2 розроблений інструментарій застосовується для проектування і реалізації алгоритмічних знань з області символьної мультиобробки.

2.1. Перехід від алгоритмів сортування до алгоритмів пошуку за допомогою інтегрованого інструментарію. Проілюструємо перехід від алгоритму паралельного сортування альтернативними вставками *САВ/А* [3] до відповідного алгоритму пошуку *АЛЬТП/П* (див. розділ 1) із використанням метаправил згортки і розгортки у розробленому ДСП-конструкторі. Основна ідея ДСП-конструктора полягає у порівневому проектуванні схем програм зверху вниз шляхом деталізації конструкцій САА-М, при цьому на кожному кроці система в діалозі з користувачем надає на вибір лише ті конструкції, підстановка яких у формований текст не порушує синтаксичну правильність схеми. Проектування алгоритму відображається у вигляді дерева конструювання, приклади якого наведено на рис. 1 і рис. 2. В алгоритмах *АЛЬТП/П* і *САВ/А* однакові стратегії мають складені оператори *АЛЬТ* та $П_i$ відповідно. Складений оператор *АЛЬТ* у паралельній схемі сортування *САВ/А* має вигляд $АЛЬТ = ЧИТ.БС(s) * ([l|\ У_1 > s]\ \{[l|\ У_1 < s]\ L(У_1)\}, \{[r|\ У_1 > s]\ P(У_1)\}) * ВСТАВ(s, У_1)$.

Дерево конструювання для наведеного оператора, побудоване у ДСП-конструкторі, зображене на рис. 1. Здійснимо згортку оператора *АЛЬТ* (заміну операторних та предикатних інтерпретацій на відповідні змінні) за системою рівностей I_1:

$$I_1 = \{v_1 = ЧИТ.БС(s),\ v_2 = l|\ У_1 > s,\ v_3 = l|\ У_1 < s,\ v_4 = L(У_1),\ v_5 = r|\ У_1 > s,\ v_6 = P(У_1),\ v_7 = ВСТАВ(s, У_1)\}.$$

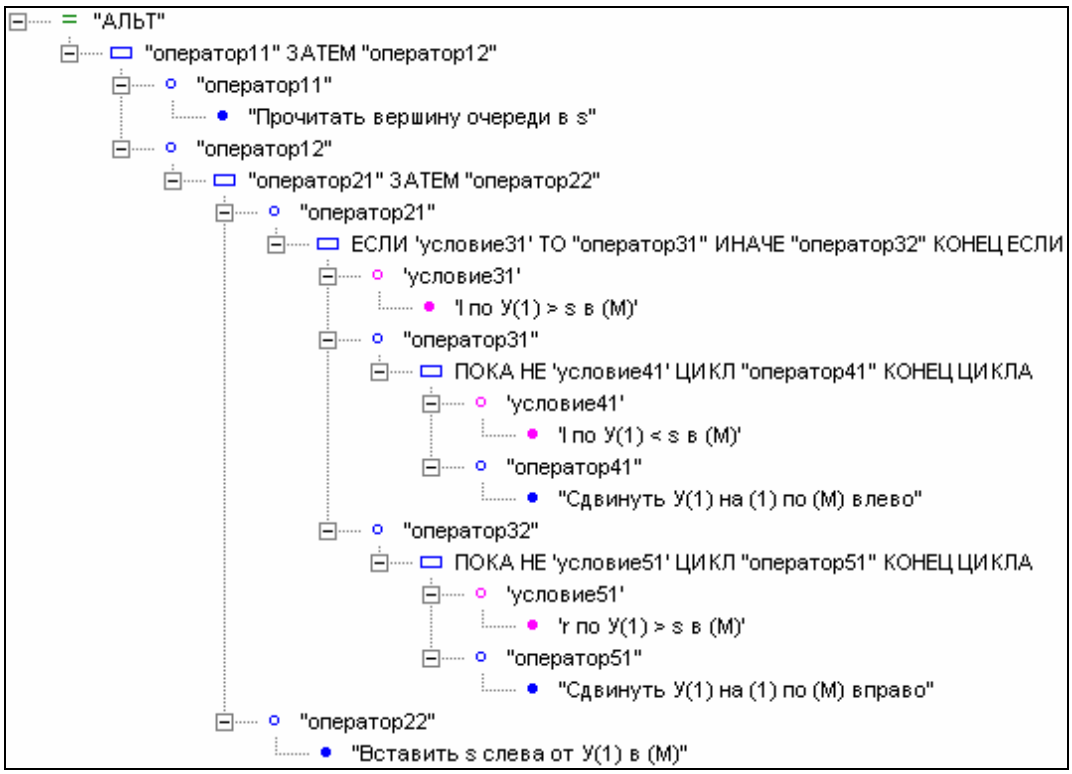

Рис. 1. Дерево конструювання для складеного оператора *АЛЬТ*, побудоване в ДСП-конструкторі

Згортка схеми *АЛЬТ* здійснюється шляхом застосування рівностей із системи I_1 у напрямку зліва направо. Результатом цієї згортки є стратегія (неінтерпретована схема) *АЛЬТС*:

$$АЛЬТС = АЛЬТ \uparrow I_1 = v_1 * ([v_2]\ \{[v_3]\ v_4\},\ \{[v_5]\ v_6\}) * v_7.$$

У ДСП-конструкторі застосування рівностей із системи I_1 реалізується видаленням вузлів дерева, які містять інтерпретацію змінних v_1, v_2, …, v_7. На рис. 1 ці вузли містять базисні оператори та предикати: “Прочитати вершину черги в *s*”, “*l* по $У(1) > s$ в (*М*)”, “*l* по $У(1) < s$ в (*М*)”, “Зсунути $У(1)$ на (1) по (*М*) вліво” і т. д. Далі для переходу до паралельного алгоритму пошуку здійснимо розгортку стратегії *АЛЬТС* (заміну змінних схеми на відповідні інтерпретації) за системою рівностей I_2:

$$I_2 = \{v_1 = УСТ\_СЕРЕД(У_i, Н, К),\ v_2 = r|\ У_i > x,\ v_3 = r|\ У_i = x,\ v_4 = L(У_i),\ v_5 = r|\ У_i = x,\ v_6 = P(У_i),\ v_7 = ([\gamma]\ A * * SUMM, NOT) * T(u_i) * ([i = m]\ S(u_1 \wedge u_2 \wedge \ldots \wedge u_{m-1}) * B, E)\}.$$

Розгортка стратегії *АЛЬТС* (застосування рівностей системи I_2 справа наліво) у ДСП-конструкторі виконується шляхом вставки у непроінтерпретовані змінні відповідних базисних та складених елементів. На рис. 2 непроінтерпретовані змінні представлені вершинами дерева: “оператор 11”, “оператор12”, “умова31”, “умова41” і т. д.

Результатом даної розгортки є складений оператор $П_i$ обробки *i*-го запиту в алгоритмі пошуку *АЛЬТП/П*:

$$П_i = АЛЬТС \downarrow I_2 = УСТ\_СЕРЕД(У_i, Н, К) * ([r|\ У_i > x]\ \{[r|\ У_i = x]\ L(У_i)\},\ \{[r|\ У_i = x]\ P(У_i)\}) * ([\gamma]\ A * * SUMM, NOT) * T(u_i) * ([i = m]\ S(u_1 \wedge u_2 \wedge \ldots \wedge u_{m-1}) * B, E).$$

Додавши до наведеної схеми $П_i$ складений оператор $\dot{\bigvee_{i=1}^{m}} П_i$, отримаємо алгоритм паралельного пошуку *АЛЬТП/П*.

2.2. Програми паралельного пошуку документів за ключовими словами. Паралельні алгоритми бульбашкового, човникового, альтернативного та бінарного пошуку (див. розділ 1) з використанням розробленого інтегрованого інструментарію було реалізовано у програмній системі пошуку Web-документів за ключовими словами [3]. Дана пошукова система складається з двох основних частин: програми-індексатора, яка здійснює підготовку інформації для подальшого пошуку, та програми, що безпосередньо здійснює пошук і видачу результатів обробки запитів користувача. Програма-індексатор переглядає Web-документи та зберігає інформацію про них у двох файлах. Перший файл містить пронумерований список усіх переглянутих документів із зазначенням їх місця розташування (адреси), назви, розміру, дати створення, опису (фрази із документа). У другому файлі (що називається індексним) зберігаються записи про слова, які містяться в документах, із зазначенням номера документа та кількості входжень слова в нього. Друга частина системи, що розглядається, шукає в індексному файлі кожне слово із запиту користувача, складає і оброблює список документів, що задовольняють запиту, і виводить їх на екран. При цьому здійснюється паралельна обробка

кількох запитів. Програмний код, що відповідає використаним алгоритмам пошуку, було синтезовано за допомогою ДСП-конструктора мовами Java та C. При реалізації паралельних програм мовою Java застосовано механізм потоків (threads) [6], а у програмах мовою C використано MPI [7].

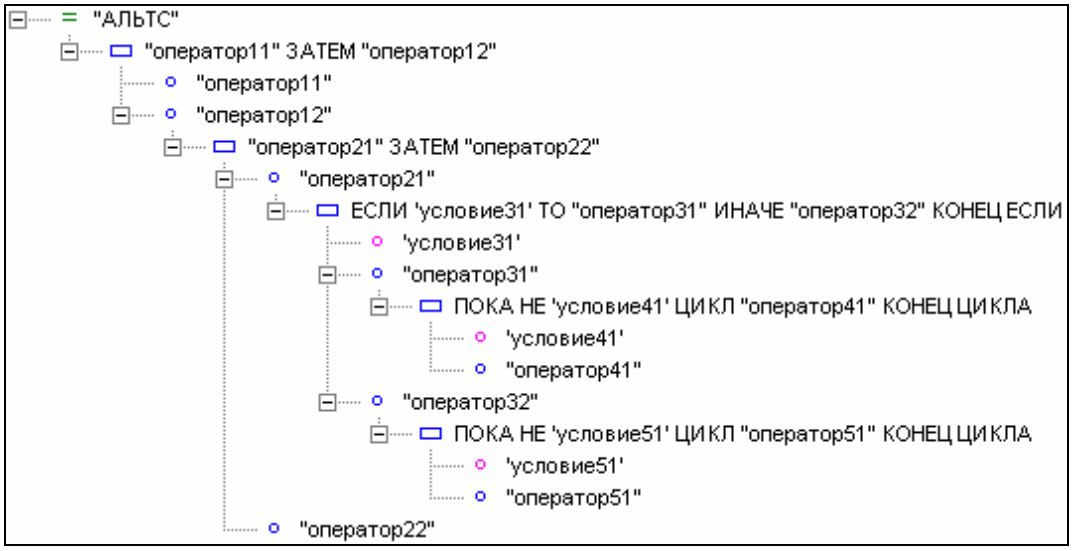

Рис. 2. Дерево конструювання для стратегії обробки *АЛЬТС*, побудоване в ДСП-конструкторі

2.2.1. Реалізація програми паралельного пошуку із застосуванням потоків. Програмне забезпечення паралельної програми пошуку документів, розробленої на основі Java, складається із таких класів:

– SearchApplet – аплет, у якому користувачу надається інтерфейс для введення запитів для пошуку документів. Інтерфейс містить кнопки додавання нових запитів, переміщення між запитами, видалення запитів та пошуку документів, що задовольняють введеним даним (рис. 3).

Рис. 3. Введення запитів у паралельній програмі пошуку

Кожний запит може складатися із кількох слів, розділених пробілами. Запуск аплета виконується із html-сторінки у Web-браузері. Введені запити передаються для обробки сервлету SearchServlet

– SearchServlet – сервлет, що виконує пошук документів. При одночасному надходженні до сервлету кількох запитів створюється відповідна кількість паралельних потоків, у кожному з яких обробляється окремий запит за допомогою методу doGet [8]. Реалізація алгоритму пошуку документів, які задовольняють *i*-му запиту (тобто складеного оператора Π_i із САА-схеми), знаходиться у методі processQuery, який викликається у методі doGet. При цьому пошук документів виконується за допомогою методів класу Searching, що розглядається нижче. Проектування алгоритму методу processQuery та генерація відповідного коду мовою Java здійснюється за допомогою ДСП-конструктора. Результати виконання запитів (список знайдених документів) передаються аплету і виводяться ним на екран у вигляді html-сторінки;

– Searching – клас, що є повторно використовуваним компонентом для розв'язання задач пошуку. Він містить опис даних, використовуваних у процесі обробки запитів: індексного файлу та файлу документів, масиву запитів, вказівників, контрольних точок, таблиці знайдених документів та ін., а також методів доступу до них.

Для проектування структури розглянутих класів, а також генерації програмного коду спільно з інтегрованим інструментарієм було застосовано систему візуалізованого проектування на основі мови UML Rational Rose 2000 [5]. На рис. 4 зображено побудовану у цій системі діаграму класів UML для пошукової системи, що розглядається.

На побудованій діаграмі класів вказано лише основні атрибути та операції класів, відношення залежності класу SearchServlet від класу Searching показано пунктирною стрілкою. За класом SearchServlet у Rational Rose було згенеровано каркасний програмний код (без реалізацій методів). Синтез коду, що реалізує методи, далі здійснено за допомогою ДСП-конструктора на основі алгоритмів виконання методів. Алгоритм паралельного

пошуку для випадку човникової обробки (паралельна схема *ЧОВНП/П* із розділу 1) подано на рис. 5 за допомогою діаграми діяльності UML.

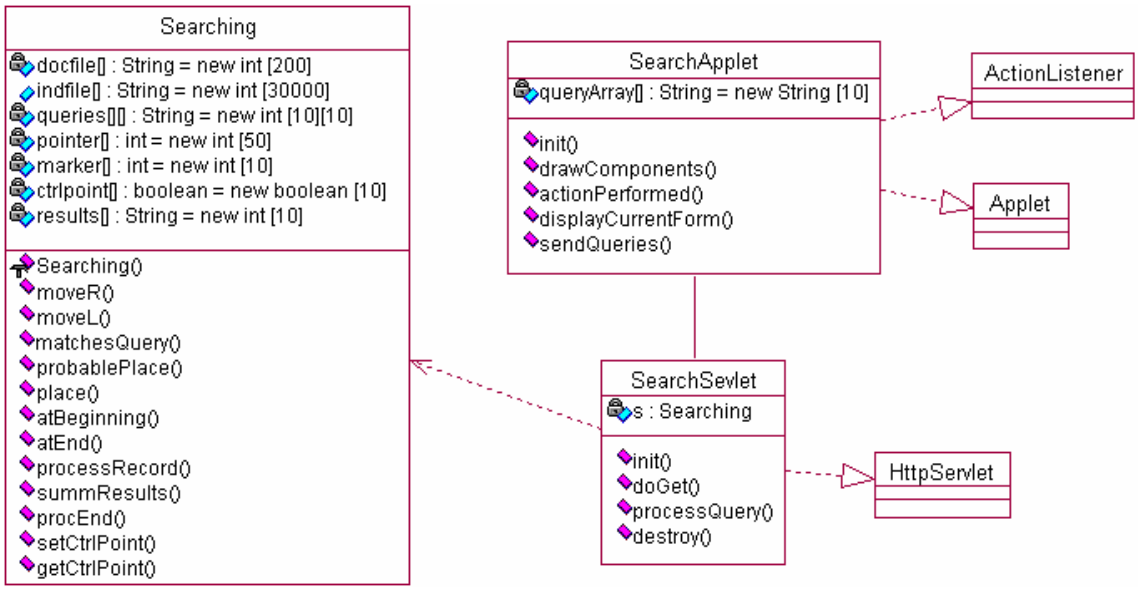

Рис. 4. Діаграма класів для паралельної програми пошуку документів за ключовими словами

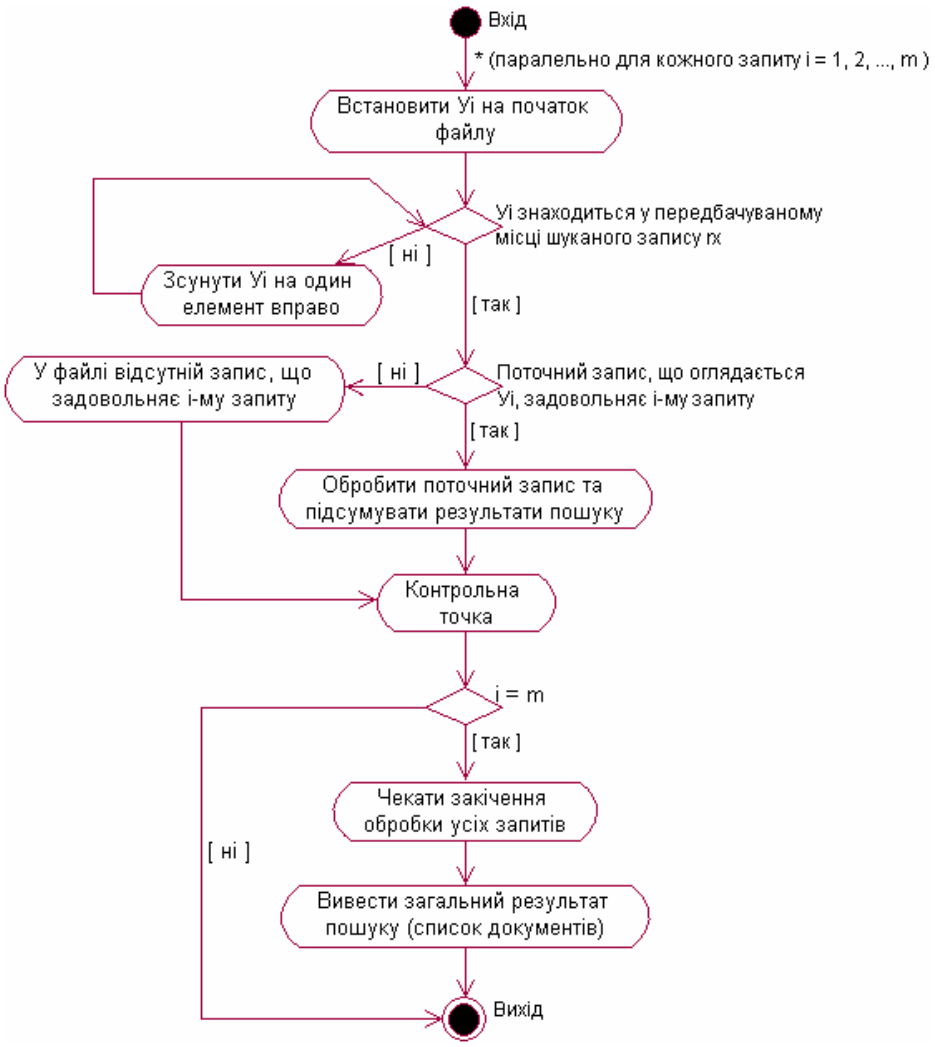

Рис. 5. Діаграма діяльності UML для паралельного алгоритму човникового пошуку записів у індексному файлі за масивом запитів

2.2.2. Реалізація програми паралельного пошуку для виконання на обчислювальному кластері. Розроблені паралельні алгоритми пошуку також було реалізовано мовою С з використанням технології MPI (Message Passing Interface, стандарт інтерфейсу передачі повідомлень). Паралельна програма в MPI є множиною процесів, кожний з яких має власний локальний адресний простір [7]. Взаємодія процесів – обмін даними і синхронізація – здійснюється за допомогою передачі повідомлень. Стандарт MPI визначає бібліотеку підпрограм, які, зокрема, реалізують операцію send, що використовується для ініціації передачі даних між двома паралельно виконуваними частинами програми, та операцію receive, призначену для вилучення цих даних із системних структур даних в область пам'яті програми. Існують також колективні операції, орієнтовані на кілька процесів. При написанні програм пошуку використано бібліотеку MPICH, яка є широко відомою реалізацією стандарту MPI.

Розроблені паралельні програми пошуку документів було апробовано у 3-процесорній кластерній системі (процесори Intel Celeron 1700 МГц, операційна система MS Windows XP). У кожній із програм пошуку здійснюється одночасна обробка 3 запитів паралельними процесами, що мають номери 0–2, де 0 відповідає основному процесу. Основний процес реалізує інтерфейс користувача, здійснює введення запитів, надсилає текст другого та третього запитів процесам 1 та 2. Далі здійснюється паралельний пошук документів, при цьому перший із запитів оброблюється процесом 0. Після закінчення обробки процеси 1 та 2 передають отримані результати (список документів) основному процесу за допомогою виклику стандартної функції MPI_Send. Процес 0 отримує ці дані за допомогою виклику функції MPI_Recv, після чого виводить сформований ним загальний список результатів паралельного пошуку.

У табл. 1 наведено час виконання розроблених паралельних програм. Відмітимо, що час виконання програм бульбашкового і човникового пошуку є приблизно однаковим. Різниця між ними виявляється лише у випадку, коли шуканий запис у вхідному файлі відсутній. Приблизно у 2 рази швидше виконується програма, реалізована за алгоритмом альтернативного пошуку. Найкращим за часом виконання є програма бінарного пошуку. У табл. 2 наведено час виконання послідовних програм по відношенню до паралельних.

Оцінимо показники продуктивності розпаралелювання обчислень у розглянутих програмах пошуку на основі обчислення коефіцієнтів прискорення $S_p = T_1/T_p$ та ефективності $E_p = S_p/p \le 1$, де – T_p час розв'язання задачі на p процесорах; T_1 – час розв'язання цієї ж задачі на одному процесорі. Як видно з табл. 2, коефіцієнт прискорення S_p програм пошуку у середньому є близьким до 2, а коефіцієнт ефективності $E_p = 2/3 \approx 0.67$.

Час виконання (у секундах) паралельних програм пошуку документів Таблиця 1

Алгоритм пошуку	Кількість процесів	Кількість записів у індексному файлі	
		15000	30000
Бульбашковий	3	0.007	0.010
Човниковий	3	0.006	0.010
Альтернативний	3	0.004	0.006
Бінарний	3	0.00079	0.00081

Час виконання послідовних програм пошуку документіввідносно паралельних Таблиця 2

Алгоритм пошуку	Кількість записів у індексному файлі	
	15000	30000
Бульбашковий	2.0	2.1
Човниковий	1.8	2.1
Альтернативний	2.0	1.9
Бінарний	1.4	3.1

3. Перспективи реалізації паралельних програм на основі інструментарію Globus Toolkit

Відзначимо перспективність використання засобів Grid-технологій при реалізації паралельних програм, подібних до розглянутих у даній статті, на кластерній мультипроцесорній архітектурі. Grid – сучасна технологія, що прогресивно розвивається, та створена для спільного використання великої кількості глобально розподілених ресурсів і підтримки високопродуктивних обчислень [9]. До інструментальних засобів Grid відноситься, зокрема, Globus Toolkit (GT), призначений для розробки сервісно-орієнтованих розподілених обчислювальних програм та інфраструктур. Його останні версії, GT3 та GT4, ґрунтуються на Grid-сервісах – розширенні Web-сервісів, які є однією з технологій розподілених обчислень поряд із CORBA, RMI, EJB. Архітектура та компоненти GT детально розглянуті у [4].

Однією з можливостей реалізації розглянутих у попередніх розділах асинхронних алгоритмів на основі інструментарію GT є розробка розподілених клієнт-серверних програм, які використовують Web-сервіси. Наприклад, розглянемо програму паралельного пошуку документів. Розробка такої програми передбачає

створення Web-сервісу, який реалізує пошук інформації про документи у базі даних відповідно переданому йому у якості параметру запиту. Для паралельної обробки копії даного Web-сервісу та бази даних розміщені на множині комп'ютерів, які у даному випадку грають роль серверів. Для опису інтерфейсу Web-сервісу застосовується мова опису Web-сервісів WSDL (Web Services Description Language). Документ WSDL є XML-документом, який визначає розташування сервісу та операції, що надаються ним. У паралельній пошуковій системі, що розглядається, програма-клієнт, розміщена на одному із комп'ютерів мережі розподіленої обробки, реалізує інтерфейс користувача, вводить запити та здійснює паралельний виклик Web-сервісів, передаючи їм запити на пошук документів. Виклик Web-сервісу передбачає посилання повідомлень між клієнтом і сервером, формат яких визначається протоколом SOAP (Simple Object Access Protocol). Клієнтська програма збирає результати пошуку, отримані Web-сервісами, та виводить загальний список знайдених документів. Слід відмітити, що Web-сервіси не залежать від платформи і мови, тому клієнтська програма може бути реалізована, наприклад, мовою C++, і виконуватися у середовищі ОС Windows, у той час як Web-сервіс реалізується мовою Java і виконується у Linux. Відзначимо можливість проектування алгоритмів та синтезу коду таких розподілених програм за допомогою розробленого інтегрованого інструментарію (див. розд. 2).

Використання Web-сервісів дозволяє здійснювати доступ до програм на віддаленому комп'ютері, що може бути досить прийнятним у багатьох випадках [10]. Однак слід відмітити, що такий підхід не задовольняє усім потребам розподіленої обробки. Розглянемо додаткові засоби GT, що дозволяють виконувати паралельні програми.

До основних компонентів системи Globus відноситься GRAM (Grid Resource Allocation and Management) – універсальний інтерфейс до засобів керування різноманітними ресурсами (процесори, пристрої зберігання, комунікації і т. д.). Компонент GRAM включає набір Web-сервісів, необхідних для ініціації, контролю, координації та керування завданнями на обчислювальних ресурсах Grid. Завдання є обчислювальними задачами, що можуть виконувати операції введення/виведення у процесі свого виконання, які впливають на стан обчислювального ресурсу і пов'язані з ним файлові системи. Специфікація завдання для виконання задається користувачем у вигляді XML-файлу опису завдання. Контроль завдання полягає у запитуванні інформації про його стан. Завдання можуть бути паралельними, тобто складатися із кількох завдань, що виконуються одночасно. Такі завдання, як правило, розміщуються на паралельному комп'ютерному обладнанні для підвищення продуктивності обчислень. Для синхронізації між паралельними процесами GRAM забезпечує механізм рандеву (rendezvous). Даний механізм означає, що програма може записати певну бінарну інформацію, наприклад дані про процес, та отримати повідомлення, коли всі інші процеси запишуть їх власну інформацію. Рандеву може бути використане у випадку, коли усім паралельним завданням необхідно зупинитися перед тим, як продовжити обчислення. Цей механізм може застосовуватися безпосередньо у коді програми або використовуватися проміжними бібліотеками, що призначені для представлення спрощеної моделі програмування для застосувань, що виконуються за допомогою GRAM. Програма managed-job-globusrun є інструментом GT4 для маніпулювання завданнями на локальному або віддаленому комп'ютері за допомогою сервісів GRAM. Для запуску паралельних завдань параметру count цієї програми присвоюється значення більше за одиницю, яке відповідає кількості копій виконуваної програми.

Із Globus Toolkit пов'язані також інструментальні засоби, що забезпечують різноманітні моделі паралельного програмування у середовищі Grid (Condor-G, DAGman, MPICH-G2, GriPhyN VDS, Nimrod-G) [10]. Серед них слід виділити MPICH-G2 [11] – реалізацію MPI, яка використовує сервіси Globus Toolkit (наприклад, що стосуються запуску завдань, безпеки) для підтримки ефективного і прозорого виконання у гетерогенних середовищах Grid. Цей інструментарій дозволяє об'єднати кілька машин, що можуть мати різні архітектури, для виконання MPI програм. MPICH-G2 автоматично перетворює дані у повідомленнях, що надсилаються між машинами різної архітектури та підтримує багатопротокольну взаємодію для передачі повідомлень. Перед запуском програми на основі MPICH-G2 користувач використовує протокол інфраструктури безпеки GSI (Grid Security Infrastructure) для своєї автентифікації на кожному вузлі розподіленої системи. Після цього для виконання MPI обчислення застосовується стандартна команда mpirun. Реалізація у MPICH-G2 цієї команди використовує мову специфікації ресурсів RSL (Resource Specification Language) для опису завдання. Користувачі пишуть RSL-сценарії, що ідентифікують ресурси (наприклад, комп'ютери) та визначають вимоги (кількість процесорів, пам'ять, час виконання) і параметри (розміщення виконуваних файлів, параметри командного рядка, змінні середовища). Ґрунтуючись на інформації у RSL-сценарії, MPICH-G2 викликає бібліотеку DUROC (Dynamically-Updated Request Online Coallocator), що постачається разом із Globus Toolkit, для розміщення та виконання програми на комп'ютерах, визначених користувачем. Ця бібліотека використовує функції API та протокол GRAM для запуску і подальшого керування обчисленнями на кожному із комп'ютерів. Після того, як процеси почали своє виконання, MPICH-G2 використовує дані із файлу RSL для створення багаторівневої кластеризації процесів, яка ґрунтується на топології мережі.

Висновки

У даній статті здійснено подальший розвиток середовища конструювання алгоритмічних знань символьної мультиобробки. Згадане середовище ґрунтується на апараті алгебри алгоритміки (стратегіях обробки, метаправилах конструювання схем) та призначене для проектування класів алгоритмів і програм.

Розпаралелено низку послідовних алгоритмів сортування і пошуку, що входять до складу створеного СКЗ. Спроектовано та реалізовано паралельні програми пошуку документів за ключовими словами із використанням розробленого інтегрованого інструментарію діалогового конструювання та синтезу синтаксично правильних об'єктно-орієнтованих програм. Генерацію програм виконано у мові Java із використанням програмних потоків, а також у мові C із застосуванням технології MPI. Експериментальна оцінка часу виконання розроблених програм пошуку на обчислювальному кластері показує хорошу степінь розпаралелюваності обчислень.

Перспективи подальших досліджень у даному напрямку пов'язані із застосуванням засобів Grid-обчислень, зокрема, інструментарію Globus Toolkit, для реалізації програм мультиобробки. При цьому важливим є подальший розвиток розробленого інтегрованого інструментарію для автоматизованого конструювання та синтезу розподілених програм для Grid-систем.

1. *Цейтлин Г.Е.* Введение в алгоритмику. – К.: Сфера, 1998. – 311 с.
2. *Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л.* Методы символьной мультиобработки. – К.: Наукова думка, 1980. – 252 с.
3. *Яценко О.А.* Розробка інтегрованих алгебро-алгоритмічних моделей: елементи теорії, інструментарій, застосування. Автореф. дис. … канд. фіз.-мат. наук. – Київ, 2005. – 17 с.
4. *Globus* Toolkit 4.0 Release Manuals. – http://www.globus.org/toolkit/docs/4.0/
5. *Фаулер М., Скотт К.* UML. Основы: Пер. с англ. – СПб.: Символ-Плюс, 2002. – 192 с.
6. *Бишоп Д.* Эффективная работа: Java 2. – К.: BHV, 2002. – 592 с.
7. *Антонов А.С.* Параллельное программирование с использованием технологии MPI: Учебное пособие. – М.: МГУ, 2004. – 71 с.
8. *Холл М., Браун Л.* Программирование для Web. Библиотека профессионала: Пер. с англ. – М.: Вильямс, 2002. – 1264 с.
9. *Foster I., Kesselman C., Tuecke S.* The Anatomy of the Grid: Enabling Scalable Virtual Organizations. – http://www.globus.org/research/papers/anatomy.pdf.
10. *GT4 Key Concepts* (A Globus Primer). – http://www.globus.org/toolkit/docs/4.0/key/GT4_Primer_0.6.pdf.
11. *Karonis N., Toonen B., Foster I.* MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. – ftp://ftp.cs.niu.edu/pub/karonis/papers/JPDC_G2/JPDC_G2.pdf.