УДК 681.3

Дерев'янченко О. В.

МОДЕЛЮВАННЯ ПАРАЛЕЛЬНИХ ПРОГРАМ ЗА ДОПОМОГОЮ СИСТЕМИ ПАРКС-JAVA

Мета праці — ознайомити з можливостями застосування паралельних обчислень за допомогою системи ПАРКС-JA VA. Ця система підтримує паралельне програмування на комп 'ютерній мережі. ПАРКС-технологія забезпечує програмні конструкції для модифікування логічної та комунікаційної структури проблеми, яка вирішується.

Вступ

На сьогодні було запропоновано багато моделей паралельного програмування, що відрізняються гнучкістю, механізмами взаємодії задач, ступенем розбивки на підзадачі, масштабованістю, модульністю. Приклади таких моделей паралельного програмування: процес/канал (потокова), обмін повідомленнями, паралелізм даних, розподілена пам'ять.

Система ПАРКС-JAVA (Паралельні Асинхронні Рекурсивно Керовані Системи) втілила кращі якості всіх згаданих моделей паралельного програмування, але є найближчою до моделі *процес/канал,* властивості якої наводимо нижче.

1. Паралельне обчислення складається з одного або більше процесів (задач), що одночасно виконуються і число яких постійне протягом часу виконання програми.

2. Задача — це послідовна програма з локальними даними. Задача має вхідні й вихідні порти, які є інтерфейсом для середовища процесу.

3. Крім звичайних операцій, задача може виконувати такі дії: надіслати повідомлення через вихідний порт, одержати повідомлення з вхідного порту, створити новий процес і завершити процес.

4. Операція, що посилається,— асинхронна, вона завершується одразу, не очікуючи отриманих даних. Операція, що виходить,— синхронна, вона блокує процес до моменту надходження повідомлення.

5. Пари з вхідного і вихідного портів з'єднуються чергами повідомлень, що їх називають *каналами*. Канали можна створювати і видаляти. Посилання на канали (порти) можна включати в повідомлення, тож зв'язність може змінитися динамічно.

6. Процеси можна розподіляти по фізичних процесорах довільно. Зауважимо, що використане відображення (розподіл) не впливає на семантику програми. Зокрема, безліч процесів можна відобразити на окремий процесор.

Виділимо особливості моделі.

Продуктивність. Абстракції послідовного профанування , такі як процедури і структури даних, ефективні тому, що вони можуть просто і раціонально відображатися на обчислювальну машину Фон Неймана. Задача і канал роблять те саме стосовно паралельного комп'ютера. Задача є частиною коду, що може виконуватися послідовно одним процесором. Якщо дві задачі, що обмінюються даними, розподілені на різні процесори, то канал зв'язку між ними відтворюється як міжпроцесорна взаємодія; а якщо на один процесор—то може використовуватися швидший механізм.

Незалежність від розподілу. Задачі взаємодіють, використовуючи той самий механізм (канали) незалежно від їх розташування, однак результат виконання програми не залежить від того, де задачі виконуються. Отже, процес проектування і реалізації алгоритмів може не стосуватися кількості процесорів, на яких алгоритм виконуватиметься; фактично, більшість реалізацій алгоритмів складаються із значно більшої кількості задач, ніж процесорів. Це пряма передумова масштабованості.

Модульність. У модульному програмуванні різні компоненти програми розробляються окремо як незалежні модулі, потім вони об'єднуються для отримання повної програми. Взаємодії між модулями обмежені добре продуманими інтерфейсами. Отже, модульність зменшує складність програм і полегшує повторне використання налагодженого коду. Задача є природним будівельним блоком. Вона інкапсулює дані і код, що працює з цими даними; порти, через які вона відсилає і приймає дані, складають її інтерфейс. Як модуль можна використовувати складнішу структуру, що реалізує певний алгоритм і має власний інтерфейс. Визначена подібність існує і в об'єктно-орієнтованій парадигмі програмування.

Детермінізм. Алгоритм або програма детерміновані, коли виконання з визначеними вхідними даними завжди має одні й ті самі результати. Попарна взаємодія в моделі задача/канал порівняно легко може гарантувати детермінізм. Кожний канал з'єднує тільки одного відправника й одного приймаючого. Задача, що приймається, блокується, поки не закінчиться операція прийому. Ці умови послаблюються тим, що в той самий вхідний порт можуть передаватися дані по різних каналах, тобто відрізних задач, але з однаковим форматом даних. Операція відсилання даних у порт також може блокуватися, якщо задача, яка приймає на іншому кінці каналу, не встигає приймати й обробляти дані.

ПАРКС-технологія програмування

ПАРКС-система [1—5] програмування — сукупність програмних засобів, що забезпечують процес розробки і реалізації алгоритмів паралельної обробки інформації. Ця система може використовуватися при створенні програмного забезпечення паралельних ЕОМ, а також логічної структури паралельних і спеціалізованих ЕОМ, що найефективніше реалізують розроблене програмне забезпечення. Подібні розробки здійснюються користувачами в звичному для них програмному середовищі, природним чином розширеним набором відповідних ПАРКС-засобів.

ПАРКС-технологія програмування є певною надбудовою над базовою платформою. Вона є набором об'єктів — структур даних та

класів, опис яких робиться базовою мовою (в нашому випадку – це JAVA) та дає змогу створювати керуючий простір (КП), що використовується для паралельного (або псевдопаралельного) виконання процесів. Важливою особливістю ПАРКС-технології є можливість динамічно (під час роботи програми) модифікувати структуру КП та керувати його діями. Таку множину об'єктів називають ПАРКС-розширенням базової мови програмування.

Основними термінами ПАРКС-технології програмування є: *точка*, *програмний канал* (ПК), *алгоритмічний модуль* (АМ).

Структура КП – це граф, вершини якого – точки КП, а ребра – ПК, що їх з'єднують. Одні й ті самі точки можуть бути з'єднані за допомогою кількох ПК різного типу. До кожної точки КП приписано АМ, що є процедурою ПАРКС-розширення базової мови (в нашому випадку – відповідний клас на JAVA). Виконання АМ може спричинити зміни у структурі КП: вилучення або створення точок, ПК, блокування виконання деяких процесів тощо. АМ виконується паралельно, однак модуль може бути затриманий до надходження певної події (умовно) або на деякий час (абсолютно). Точки КП можуть містити керуючі підпростори будь-якої глибини, тобто КП може мати будь-який рівень рекурсії. Точки КП, з'єднані за допомогою ПК, можуть надсилати та отримувати повідомлення.

Система ПАРКС-JAVA

Система ПАРКС-JAVA [6–8] реалізується на основі локальної чи глобальної комп'ютерної мережі, надаючи можливість користувачам малопотужних комп'ютерів використовувати ресурси мультипроцесорних комплексів або комп'ютерної мережі. В обох випадках програмне забезпечення підтримує процеси взаємодії різних вузлів мережі між собою на основі обміну повідомленнями по мережі.

Розглянемо множину операторів паралельного програмування, яка реалізує ПАРКС-технологію програмування та надає можливість розширити базову мову JAVA до паралельної. Система має кілька модулів.

Головний модуль системи – parcs. Він містить реалізацію функцій системи ПАРКС-JAVA, які беруть участь в обчисленні. Основні компоненти системи:

клас parcs.task – об'єкт задача

public boolean addJarFile(String filename) – додає jar-файл у список файлів, що завантажуються для виконання АМ, де filename – ім'я файла на диску, повертає true – коли файл ще не був завантажений;

public void end() – завершення задачі;

клас parcs.point – об'єкт точка

public point(task curtask) – створення нової точки;

public channel createChannel() – створення каналу, що з'єднує поточну точку (у якій виконується код) із даною точкою, функція повертає посилання на канал;

public channel createChannel(point p, point q) – створення каналу, що з'єднує точку *p* з точкою *q*, якщо такого каналу не існує, функція повертає посилання на канал;

public channel chan = null – посилання на канал зв'язку з точкою;

public task curTask – задача, до якої приписана точка;

public HostInfo host = null – інформація про машину, до якої приписана точка;

public void execute(String AMname) – запускає в точці АМ, клас пересилається потрібній точці і запускається на виконання його метод run. Клас, ім'я якого передається як параметр функції, повинен реалізовувати інтерфейс АМ, що містить єдиний метод –

public void run(task curtask, channel parent) – починає виконання АМ, де curtask – задача, parent – посилання на канал, що зв'язаний з батьківською точкою;

клас parcs.channel – об'єкт канал

public void write(byte x) – відіслати в канал змінну *x* типу byte;

public void write(int x) – відіслати в канал змінну *x* типу int;

public void write(long x) – відіслати в канал змінну *x* типу long;

public void write(double x) – відіслати в канал змінну *x* типу double;

public void write(Serializable obj) – відіслати об'єкт, який реалізує інтерфейс Serializable, тобто розроблений з можливістю запису в канал (більшість стандартних класів мови Java2, що зберігають дані, реалізують цей інтерфейс);

public void write(*byte*[] *buf*) – відіслати в канал масив байтів;

public byte readByte() – отримати з каналу число типу byte;

public int readInt() – отримати з каналу число типу int;

public long readLong() – отримати з каналу число типу long;

public double readDouble() – отримати з каналу число типу double;

public Object readObject() – отримати з каналу об'єкт;

public void read(byte[] *buf*) – отримати з каналу масив байтів та записати його.

Центральну роль у такій реалізації відіграє процес, що відпрацьовує основні функції КП: створює точки та канали, приписує АМ точкам, підтримує взаємодію між точками під час роботи. Умовна його назва – *демон* (клас Daemon). Спілкування з ним проходить або через локальний файл, або через потік вводу-виводу TCP/IP. *Демон* має бути запущений на кожному комп'ютері мережі, який братиме участь в обчисленнях. *Демон* постійно міститься в пам'яті і приймає з'єднання з мережі на TCP-порт. При надходженні запиту на виконання АМ він породжує новий Java-потік, в якому запускає модуль на виконання. У процесі виконання початкового АМ Демон завантажує з фізичного диску програмний код інших алгоритмічних модулів.

Приклади побудови паралельної програми в середовищі ПАРКС-JAVA

Розглянемо можливості побудови різних моделей паралельних програм на мові ПАРКС-JAVA. Спочатку на прикладі класичної задачі множення матриць великої розмірності [9] змоделюємо динамічну структуру у вигляді дерева. Необхідно помножити дві матриці великої розмірності $A\,(n \times k)$ і $B(k \times m)$. Потрібно здійснити розбиття задачі на задачі меншої розмірності. Кожен АМ здійснюватиме дії над матрицями розмірності d, де $d << n, k, m$. Задачу $C = A \times B$ можна звести до поблокових дій:

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21};\ C_{12} = A_{11} \times B_{12} + A_{12} \times B_{22};\ (1)$$

$$C_{21} = A_{21} \times B_{11} + A_{22} \times B_{21};\ C_{22} = A_{21} \times B_{12} + A_{22} \times B_{22};$$

де $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix},$

$$C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}.$$

Якщо отримані блоки не досягнули потрібної розмірності, то процедуру можна продовжити. Як КП обираємо 8-дерево, фрагмент якого зображений на рис. 1, у кожній вершині якого виконуватиметься відповідний АМ (множення матриць у нашому випадку). Для спрощення розглянемо випадок, коли кількість блоків відповідає рис. 1, тобто ми розбиватимемо кожну матрицю тільки на чотири частини (фіксуємо значення *d)*.

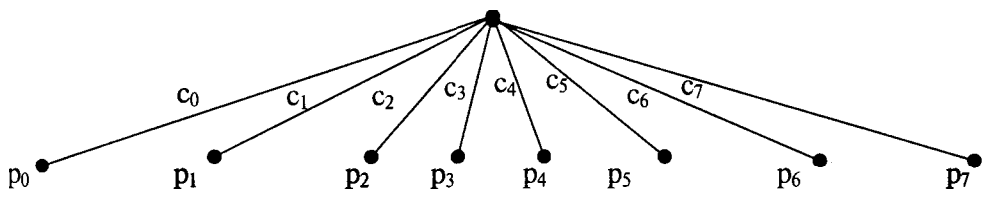
Наприклад, координати блоків матриці $A(n \times k)$ визначаються так:

$$A = \begin{pmatrix} A_{11}(0,0,n/2,k/2) \\ A_{21}(n/2,0,n/2+n1,k/2) \\ A_{12}(0,k/2,n/2,k/2+k1) \\ A_{22}(n/2,k/2,n/2+n1,k/2+k1) \end{pmatrix},$$

де $n1 = n\%2$, $k1 = k\%2$.

При реалізації паралельного алгоритму необхідно створити набір класів (АМ), які реалізують інтерфейс АМ. Кожний АМ запускається в окремій точці методом point.execute(String). Метод run запускається на виконання *демоном,* якщо надійшли відповідні команди, де curtask – задача, що виконуватиметься, а parent – канал, що з'єднується з батьківською точкою, або null, коли АМ початковий. Покажемо частину коду на мові JAVA, яка реалізує паралельні дії нашого алгоритму (не розкриваючи додаткових методів, назви яких виявлять дії, які вони виконують):

```
public class Matrixs implements AM{
 public static void main(String[] args) {
```



(1,1)(1,1) (1,2)(2,1) (1,1)(1,2) (1,2)(2,2) (2,1)(1,1) (2,2)(2,1) (2,1)(1,2) (2,2)(2,2)

Рис. 1. Представлення КП - «ДЕРЕВО» відповідно до формул (1)

```
        Const.parseArgs(args);
        task curtask = new task();
```

//Завантажуємо всі методи, що реалізують наш алгоритм;

```
    curtask.addJarFile("Matrixes.jar");
        (new Matrixs()).run(curtask, (channel)null);
        curtask.end();
    }
    public void run(task curtask, channel parent) {
        Matrix A = new Matrix(curtask.findFile
("Matrix1.data"));
        Matrix B = new Matrix(curtask.findFile
("Matrix2.data"));
    //Формулюємо представлення КП згідно з рис. 1;
        point[] p = new point[8];
        channel[] c = new channel[8];
        for (int i = 0; i < 8; i++) {
                p[i] = curtask.createPoint();
                c[i] = p[i].createChannel();
                p[i].execute("Matr");
        }
```

// Представлення блоків матриць з формулами (1) та розподіл по каналах:

```
    c[0].write(A.SubMatrix(0, 0, A.Height() / 2,
A.Width() / 2));
    c[0].write(B.SubMatrix(0, 0, B.Height() / 2,
B.Width() / 2));
    c[1].write(A.SubMatrix(0, A.Width() / 2,
A.Height() / 2, A.Width() / 2 + A.Width() % 2));
    c[1].write(B.SubMatrix(B.Height() / 2, 0,
B.Height() / 2 + B.Height() % 2, B.Width() / 2));
    c[2].write(A.SubMatrix(0, 0, A.Height() / 2,
A.Width() / 2));
    c[2].write(B.SubMatrix(0, B.Width() / 2,
B.Height() / 2, B.Width() / 2 + B.Width() % 2));
    c[3].write(A.SubMatrix(0, A.Width() / 2,
A.Height() / 2, A.Width() / 2 + A.Width() % 2));
    c[3].write(B.SubMatrix(B.Height() / 2,
B.Width() / 2, B.Height() / 2 + B.Height() % 2,
B.Width() / 2 + B.Width() % 2));
    c[4].write(A.SubMatrix(A.Height() / 2, 0,
A.Height() / 2 + A.Height() % 2, A.Width() / 2));
    c[4].write(B.SubMatrix(0, 0, B.Height() / 2,
B.Width() / 2));
    c[5].write(A.SubMatrix(A.Height() / 2, A.Width()
/ 2, A.Height() / 2 + A.Height() % 2, A.Width() / 2 +
A.Width() % 2));
    c[5].write(B.SubMatrix(B.Height() / 2, 0,
B.Height() / 2 + B.Height() % 2, B.Width() / 2));
    c[6].write(A.SubMatrix(A.Height() / 2, 0,
A.Height() / 2 + A.Height() % 2, A.Width() / 2));
    c[6].write(B.SubMatrix(0, B.Width() / 2,
B.Height() / 2, B.Width() / 2 + B.Width() % 2));
    c[7].write(A.SubMatrix(A.Height() / 2, A.Width()
/ 2, A.Height() / 2 + A.Height() % 2, A.Width() / 2 +
A.Width() % 2));
    c[7].write(B.SubMatrix(B.Height() / 2,
B.Width() / 2 , B.Height() / 2 + B.Height() % 2,
B.Width()/ 2 + B.Width() % 2));
```

//Додавання відповідних блоків матриць та прийняття даних із каналів:

```
    Matrix[][] Parts = new Matrix[2][2];
    Parts[0][0] = (Matrix)c[0].readObject();
    Parts[0][0].Add((Matrix)c[1].readObject());
    Parts[0][1] = (Matrix)c[2].readObject();
    Parts[0][1].Add((Matrix)c[3].readObject());
    Parts[1][0] = (Matrix)c[4].readObject();
    Parts[1][0].Add((Matrix)c[5].readObject());
    Parts[1][1] = (Matrix)c[6].readObject();
    Parts[1][1].Add((Matrix)c[7].readObject());
```

//Заповнення матриць:

```
    Matrix Res = new Matrix(A.Height(), B.Width());
    Res.FillSubMatrix(Parts[0][0], 0, 0);
    Res.FillSubMatrix(Parts[0][1], 0, Res.Width() /
2);
    Res.FillSubMatrix(Parts[1][0], Res.Height() / 2, 0);
    Res.FillSubMatrix(Parts[1][1], Res.Height() / 2,
Res.Width() / 2);
    System.out.println("\nResult found. Saving to file
Matrix.res");
    Res.Save(curtask.addPath("Matrix.res"));
        }
    }

    public class Matr implements AM{
        public void run(task curtask, channel parent) {
        Matrix m = (Matrix)parent.readObject();
        Matrix m1 = (Matrix)parent.readObject();
        Matrix res = new Matrix(m.Height(),
m1.Width());
```

//Функції представлення та множення блоків матриць:

```
        res.Assign(m);
        res.MultiplyBy(m1);
```

//Передаємо результат у канал:

```
        parent.write(res);
    }
    }
```

Розглянемо приклади побудови статичних моделей КП. Подамо тільки фрагменти коду, що відповідають за утворення КП. Спочатку роз-

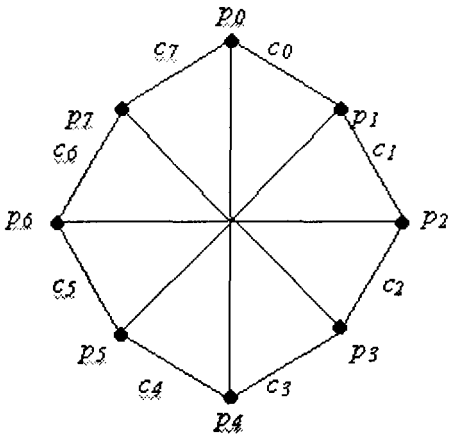


Рис. 2. Представлення КП моделі «КІЛЬЦЕ»

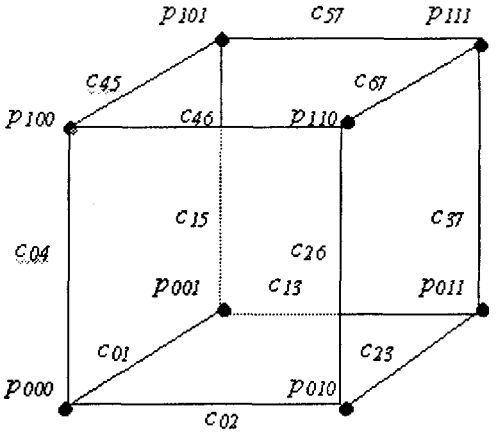


Рис. 3. Представлення КП моделі «3-КУБ»

глянемо модель «КІЛЬЦЕ» (рис. 2), а потім модель «3-КУБ » (рис. 3).

```
//Формуємо представлення КП моделі «КІЛЬЦЕ» (рис. 2):
point[] p = new point[n];
channel[] c = new channel[n];
for (int i = 0; i < n; i++) p[i] = curtask.createPoint();
for (int i = 0; i < n-1; i++) c[i] = createChannel(p[i], p[i+1]);
c[n – 1] = createChannel(p[n – 1], p[0]);
//Запускаємо відповідні АМ:
for (int i = 0; i < n; i++) p[i].execute ( ў
"TASK_AM" + i);

//Формуємо представлення КП моделі «3-КУБ» (рис. 3):
point[][][] p = new point[2][2][2];
    channel[][][] c = new channel[8][8];
    for (int i = 0; i < 2; i++)
    for (int j = 0; j < 2; j++)
for (int k = 0; k < 2; k++)
            p[i][j][k] = curtask.createPoint();
    for (int i = 0; i < 2; i++)
    for (int j = 0; j < 2; j++)
for (int k = 0; k < 2; k++) {
    int pi = i * 4 + j * 2 + k;
int pj = ((i + 1) % 2) * 4 + j * 2 + k;
    if ( c[pi][pj] == null )
            c[pi][pj] = c[pj][pi] = createChannel(p[i][j][k], p[(i + 1)%2][j][k]);
    pj = i * 4 + ((j + 1) % 2)*2 + k;
 if (c[pi][pj] == null )
c[pi][pj] = c[pj][pi] = createChannel(p[i][j][k], p[i][(j+ 1) % 2][k]);
    pj = i * 4 + j * 2+(k+1) % 2;
 if (c[pi][pj] == null)
c[pi][pj] = c[pj][pi] = createChannel(p[i][j][k], p[i][j][(k + 1) % 2]);
}
//Запускаємо відповідні АМ:
    for (int i = 0; i < 2; i++)
    for (int j = 0; j < 2; j++)
for (int k = 0; k < 2; k++)
p[i] [j] [k].execute("TASK_AM" + i + j + k).
```

Висновки

Система ПАРКС-JAVA забезпечує просту методологію написання паралельних програм. Користувачеві потрібно лише, застосовуючи методи системи, описувати предметне середовище і моделювати його за допомогою КП. Усі класи (АМ), які були створені при реалізації попередніх паралельних алгоритмів, можливо використовувати при формуванні наступних паралельних програм. Таким чином формується база класів АМ.

Під час роботи з системою ми можемо моделювати реальні обчислювальні задачі за допомогою побудови динамічної чи статичної структури керуючого простору та різних його конфігурацій (дерево, кільце, куб та ін.). Виникає проблема оптимального керування паралельними обчислювальними процесами, яка є однією з найскладніших в паралельних обчисленнях. Ефективність роботи паралельної обчислювальної системи ПАРКС-JAVA залежить не тільки від її конкретної структури, а й від того, як реалізується її виконання на конкретній обчислювальній системі, і від багатьох зовнішніх факторів.

Реалізація цієї системи може використовуватися для розв'язку широкого класу задач, де потрібне ефективне застосування паралельного програмування.

1. *Глушков В. М., Анисимов А. В.* Управляющие пространства в асинхронных параллельных вычислениях // Кибернетика.-1980.- № 5.- С. 1-9.
2. *Анисимов А. В., Борейша Ю.* £., *Карапетян М. С.* Параллельное программирование экономических систем на основе декомпозиционных методов // Кибернетика.— 1990.-№ 3.-С. 105-110.
3. *Анисимов А. В., Кулябко П. П.* Программирование параллельных процессов в управляющих пространствах // Кибернетика,- 1984.- № 3,- С. 79-88.
4. *Анисимов А. В., Кулябко П. П.* Особенности ПАРУС-технологии // Кибернетика и системный анализ.— 1993.— № 3.- С. 128-137.
5. *Анисимов А. В., Кулябко П. П.* Моделирование сети Петри с помощью ПАРУС-средств // Проблемы программирования,- 1997,— Вып. 2.— С. 45—56.
6. *Анисимов А. В., Деревянченко А. В.* Построение виртуального параллельного пространства с использованием технологии ПАРУС-JAVA // Материалы Международной научно-технической конференции ИМС2003.— Т. 2.— С.18-19.
7. *Анісімов А. В., Дерев'янченко О. В.* Система ПАРКС-JAVA як засіб вирішення паралельних алгоритмів на комп'ютерній мережі // Проблемы программирования. Материалы IV международной научно-практической конференции УкрПРОГ'2004.- 2004.- № 2-3.- С. 282-284.
8. *Анисимов А. В., Деревянченко А. В., Литвинов Д. В.* Архитектура системы для параллельных вычислений ПАРУС-JAVA // Материалы Международной научно-технической конференции ИМС2004.- Т. 1.— С. 132—135.
9. *Дерев'янченко О. В.* Побудова паралельних програм за допомогою системи ПАРКС-JAVA // Матеріали Міжнародної конференції «Теоретичні та прикладні аспекти побудови програмних систем»,— Київ, 5-8 жовтня 2004 р.— С. 313-320.

O. V. Derevyanchenko

MODELING OF PARALLEL PROGRAMS WITH PARCS-JAVA SYSTEM

The aim of the paper is an acquaintance with parallel computations based on PARCS-JAVA system. It's used for supporting of parallel programming in computing network. PARCS-technologies provide software construction and modification of logical and communication structure of a solving problem.