Паралельні обчислення в алгоритмах імітації

Прискорення експериментальних досліджень

- кілька прогонів однакової моделі виконують одночасно для отримання більш точного результату (див. тактичне планування експериментів):
  - прогони виконуються в окремих потоках, основний метод очікує завершення роботи всіх потоків, потім виконує обчислення середнього значення
- кілька моделей з різними параметрами виконують одночасно (див. стратегічне планування експериментів, оптимізація еволюційними методами):
  - основний метод здійснює обчислення параметрів, при яких виконується моделювання, та обробку результатів імітації
  - імітація моделей для різних наборів параметрів здійснюється в окремих потоках

Графічний інтерфейс

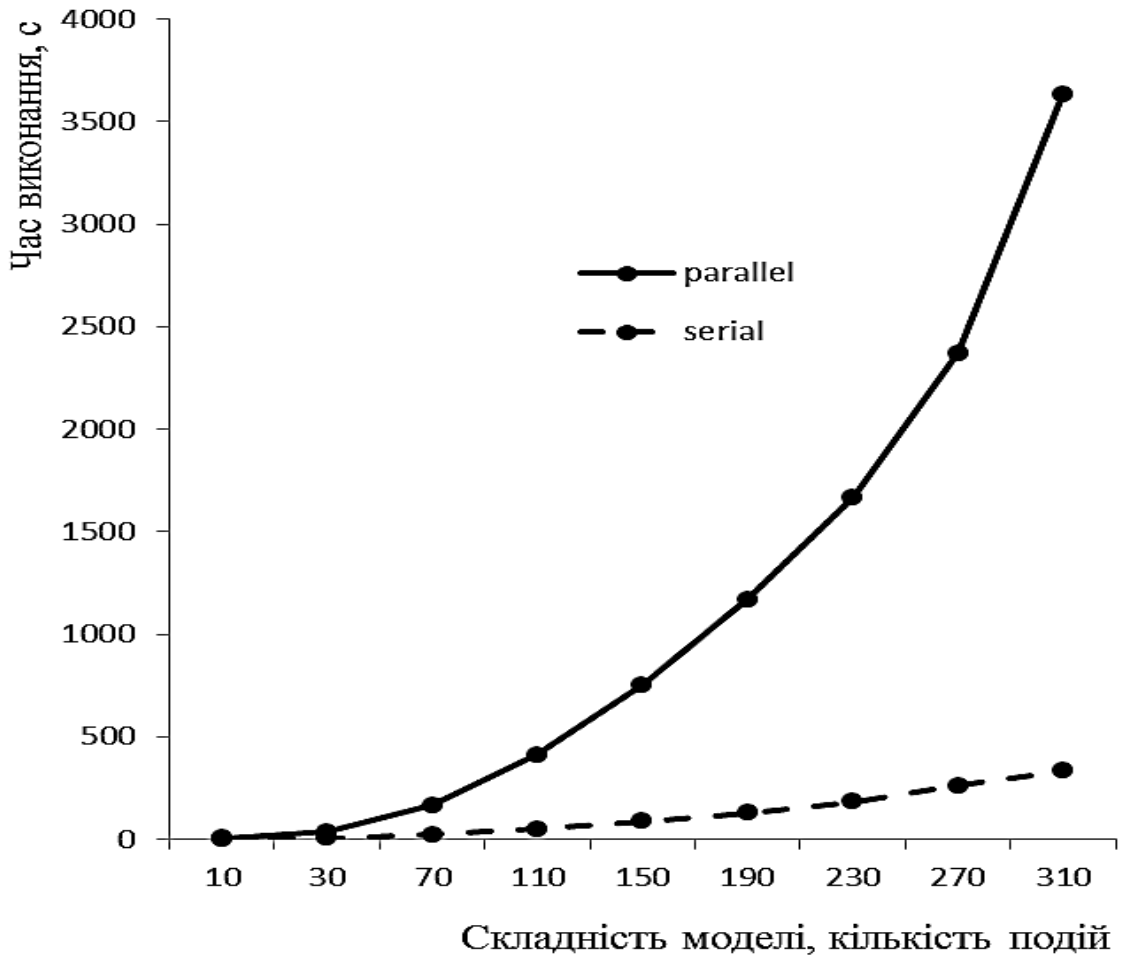
- Динамічне відтворення результатів моделювання одночасно з обчисленням моделі:
  - в одному потоці відбувається імітація, інший з затримкою виводить поточні значення вихідних характеристик
- Анімація моделювання одночасно з обчисленням моделі:
  - перемальовування елементів моделі виконується з затримкою

Прискорення виконання алгоритму імітації

Варіанти розпаралелювання:

- Відшукати незалежні частини моделі, що можна виконувати одночасно
  - Проте виявлення таких частин є самостійною складною задачею
- Виконувати розпаралелювання в межах операцій, що часто повторюються. Наприклад, для мережі Петрі можна виконувати одночасно перевірку запуску переходів та вхід маркерів в переходи
  - Накладні витрати на створення потоків (або задач) перевищують виграш в часі, отриманий за рахунок одночасного виконання задач
- Виконувати імітацію окремих фрагментів моделі одночасно
  - Через використання спільної змінної часу алгоритм тільки сповільнюється

Ефективність паралельного алгоритму імітації у випадку глобальної змінної часу



Прискорення виконання алгоритму імітації

Варіант розпаралелювання:

- Виконувати імітацію окремих фрагментів моделі одночасно не використовуючи спільну змінну часу

Варіант розпаралелювання Петрі-об’єктної моделі:

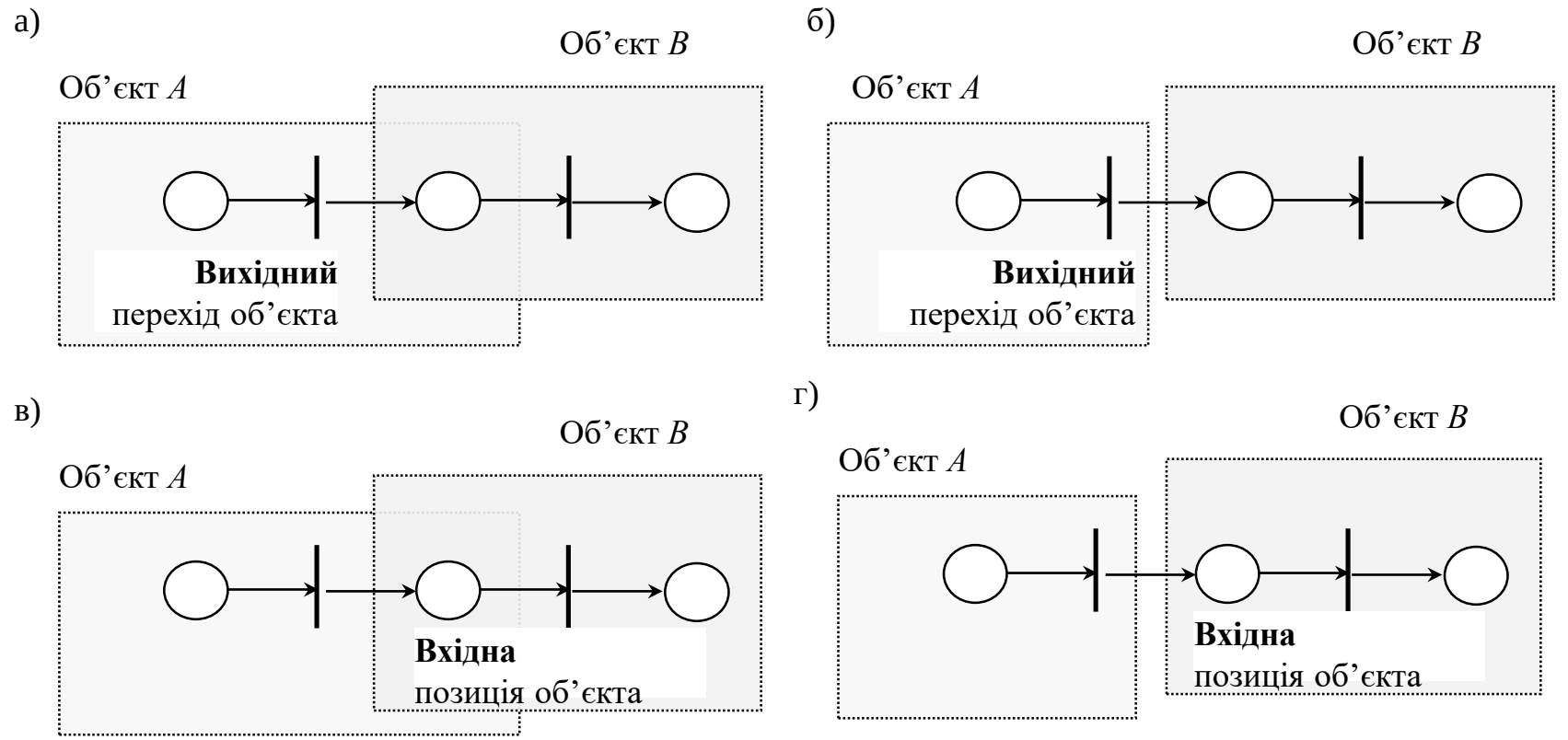
- Фрагменти моделі – це Петрі-об’єкти, з яких вона складається
- Кожний Петрі-об’єкт відтворює своє функціонування з урахуванням подій в Петрі-об’єктах, з якими він пов’язаний.
- Інформація про події надходить від об’єктів і розглядається як зовнішня

Паралельний алгоритм імітації Петрі-об’єктної моделі

Означення:

- Перехід мережі Петрі-об’єкта є вихідним, якщо при його запуску здійснюється вихід у спільну позицію з іншим об’єктом або здійснюється вихід у позицію іншого об’єкта (рис.1 а,б).
- Позиція об’єкта є вхідною, якщо вона є спільною з позицією іншого об’єкта або у неї здійснюється вихід з переходу іншого об’єкта (рис.1 в,г).
- Якщо об’єкт *A* має вихідний перехід, з якого здійснюється вихід маркерів у позицію об’єкта *B*, то об’єкт *B* є next-об’єктом для об’єкта *A*
- Якщо об’єкт B має позицію, в яку здійснюється вихід маркерів з вихідного переходу об’єкта *A*, то об’єкт *A* є previous-об’єктом для об’єкта *B*.
- Безпечним інтервалом імітації об’єкта є інтервал часу між двома послідовними виходами маркерів з вихідного переходу його previous-об’єкта.

Взаємодія Петрі-об’єктів



а) вихід маркерів з переходу об’єкта у спільну позицію його next-об’єкта, б) вихід маркерів з переходу об’єкта у позицію його next-об’єкта, в) вхід маркерів у спільну позицію об’єкта з його previous-об’єкта, г) вхід маркерів у позицію об’єкта з переходу його previous-об’єкта.

Правила паралельного функціонування Петрі-об’єктів

1. Кожний Петрі-об’єкт здійснює імітацію в локальному часі в окремому потоці.

2. Кожний об’єкт, який має next-об’єкт, передає йому інформацію про моменти виходу маркерів з об’єкта і призупиняє своє функціонування при значному накопиченні такої інформації.

3. Кожен об’єкт, який має previous-об’єкт, здійснює імітацію в межах до наступної події входу в нього маркерів з previous-об’єкта, поступово просуваючи свій локальний час до повного вичерпання накопиченої інформації про вхідні події, або очікує надходження інформації про вхідні події зі свого previous-об’єкта.

4. Об’єкт, який має previous-об’єкт, при досягненні моменту часу входу маркерів в об’єкт, відновлює вихід маркерів у спільну позицію з переходу previous-об’єкта та продовжує імітацію.

5. Об’єкт, який має next-об’єкт, при досягненні моменту виходу маркерів з об’єкта, призупиняє вихід маркерів у позицію next-об’єкта. Цей вихід відновить next-об’єкт у відповідний момент часу свого функціонування.

6. Об’єкт, який вичерпав час моделювання, передає повідомлення про це next-об’єкту, якщо такий є, і завершує свою роботу. Разом з завершенням роботи об’єкта завершує роботу і потік, ним генерований.

7. Об’єкт, який отримав від previous-об’єкта сигнал про завершення його роботи і, водночас, вичерпав усі накопичені події, припиняє свою роботу.

Буфер зовнішніх подій

- зберігає моменти часу надходження маркерів з інших об’єктів, що не опрацьовані на поточний момент часу
- в межах від одного моменту зовнішньої події до наступної гарантовано не виникає зовнішніх подій, отже, об’єкт може виконувати імітацію своїх внутрішніх подій
- порожній стан буфера означає, що наступний момент зовнішньої події невідомий (на поточний момент виконання алгоритму імітації)
- стан буфера, в якому останній його елемент є нескінченність (максимально допустиме число), означає, що previous-об’єкт завершив свою роботу і надходження зовнішніх подій не очікується.

Локальний час об’єкта просувається за такими правилами

Локальний час об’єкта просувається за такими правилами:

- якщо час найближчої події менший за межу безпечного інтервалу, то просунути локальний час у момент найближчої події, виконати (внутрішні) події, для яких час збігається з поточним моментом, та продовжити імітацію об’єкта;
- інакше
  - якщо межа безпечного інтервалу менша за межу інтервалу моделювання
    - просунути локальний час на межу безпечного інтервалу та виконати зовнішню подію (відновлення маркерів у вхідних позиціях),
  - інакше просунути локальний час на межу безпечного інтервалу та завершити імітацію об’єкта

Межа безпечного інтервалу

Межа безпечного інтервалу визначається за наступними правилами:

- якщо немає previous-об’єкта, то межа встановлюється в час моделювання;
- інакше
  - якщо буфер зовнішніх подій не порожній і перша подія менша за час моделювання, то межа встановлюється в момент першої зовнішньої події;
  - інакше - в час моделювання.

Об’єкт

Об’єкт знаходиться в очікуванні (призупиняє своє функціонування), якщо і тільки якщо:

- він знаходиться у stop-стані: кількість маркерів недостатня для запуску переходів і момент виходу маркерів з переходу більший за границю безпечного інтервалу і, якщо є previous-об’єкт, перша подія в буфері менша за границю безпечного інтервалу;
- є previous-об’єкт і буфер зовнішніх подій порожній;
- є next-об’єкт і його буфер зовнішніх подій перевищує заданий ліміт.

Об’єкт розблоковується (припиняє очікування) якщо і тільки якщо:

- його previous-об’єкт здійснив вихід у позицію, спільну з даним об’єктом;
- його next-об’єкт подав сигнал про відновлення роботи об’єкта у разі, якщо він вичерпав буфер подій до заданого ліміту.

Об’єкт завершує імітацію, якщо досягнутий кінець інтервалу моделювання.

Синхронізація виходу маркерів у спільну позицію

Синхронізація виходу маркерів у спільну позицію відбувається таким чином:

- у момент здійснення виходу маркерів з вихідного переходу Петрі-об’єкта вихід маркерів у позицію next-об’єкта не здійснюється, замість цього поточний момент часу запам’ятовується в буфері зовнішніх подій next-об’єкта;
- коли локальний час об’єкта досягає першого значення буфера подій, здійснюється вхід маркерів у вхідну позицію об’єкта, а перше значення з буфера зовнішніх подій видаляється.

Умова завершення роботи потоків

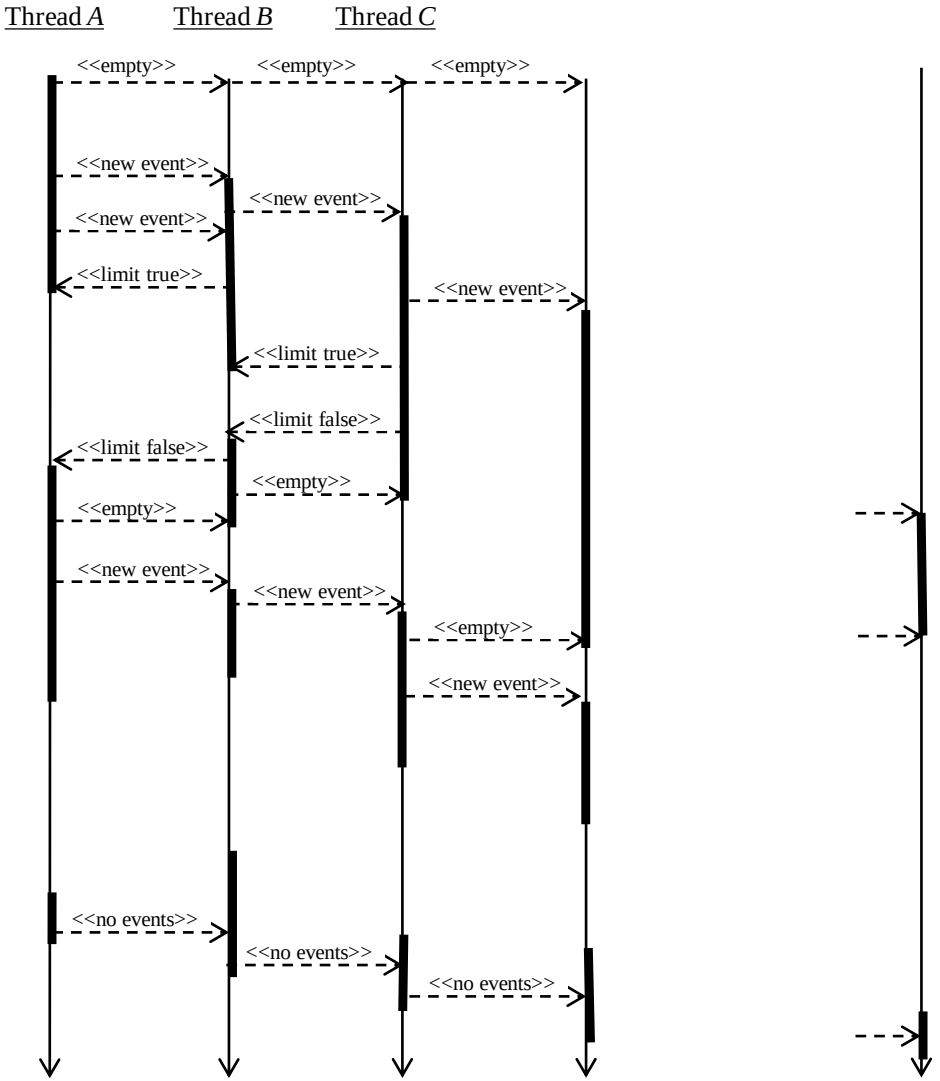
Якщо в буфері об’єкта перший елемент нескінченність, це означає, що його previous-об’єкт завершив імітацію на інтервалі імітації і зовнішніх подій більше не очікується.

Об’єкт продовжує імітацію до завершення інтервалу моделювання без очікування зовнішніх подій.

Алгоритм, описаний вище, виходить з таких припущень: модель має хоча б один об’єкт, для якого не заданий previous-об’єкт, і хоча б один об’єкт, для якого не заданий next-об’єкт. В реальних умовах це відповідає відкритій системі.

- Якщо об’єкт має декілька previous-об’єктів, то для кожного з них створюється свій буфер подій. Момент найближчої події можна визначити тільки, коли надійшла інформація про наступні події від усіх previous-об’єктів. Тому робота такого об’єкта призупиняється, якщо хоч один буфер подій порожній, і відновлюється, якщо усі буфери подій не порожні. Щоб запобігти взаємному блокуванню об’єктів, потрібно відслідковувати стан, в якому об’єкт і його previous-об’єкти одночасно знаходяться в стані очікування (коли хоч один з їх буферів подій є порожнім). В цьому випадку потрібно визначити найближчу подію для об’єкта і його previous-об’єктів, просунути локальний час в момент найближчої події і продовжити моделювання.
- Якщо об’єкт має декілька next-об’єктів, то робота такого об’єкта призупиняється, якщо досягає ліміту буфер подій хоч в одному з його next-об’єктів, і відновлюється, якщо в усіх next-об’єктах кількість подій в буфері подій менша за назначений ліміт.

Взаємодія потоків паралельного алгоритму імітації Петрі-об’єктної моделі



Потоки *A*, *B*, *C* ілюструють дію потоків, що запускають імітацію Петрі-об’єктів *A*, *B*, *C* таких, що:

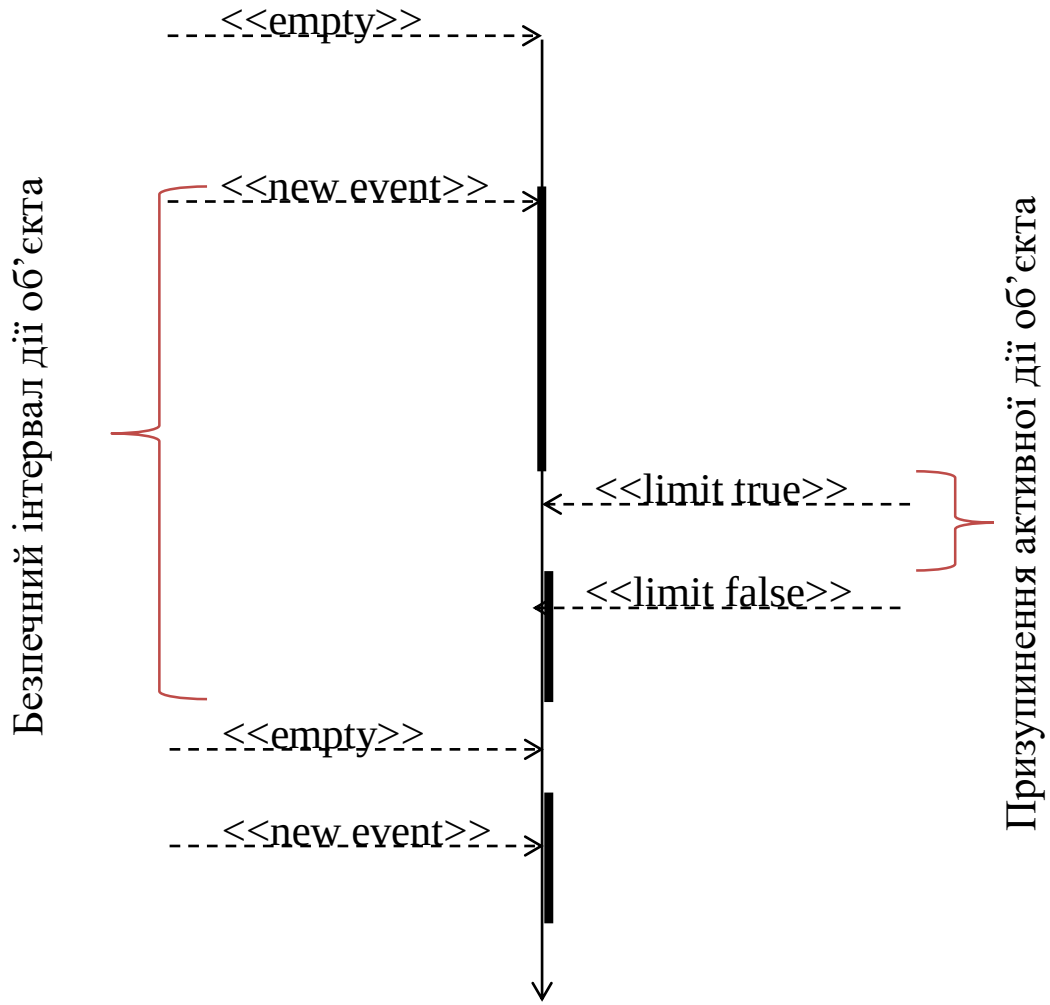
- *A* є previous-об’єктом для *B*,
- *B* є next-об’єктом для *A* і previous-об’єктом для *C*,
- *C* є next-об’єктом для *B*.

Пара повідомлень <<new event>> та <<empty>> інформує об’єкт про надходження нової події від previous-об’єкта та, навпаки, що таких подій немає.

Повідомлення <<no event>> сповіщає next-об’єкт про завершення імітації об’єктом, а значить про припинення надходження подій від previous-об’єкта.

Пара повідомлень <<limit true>>, <<limit false>> інформує об’єкт про перевищення ліміту буферу зовнішніх подій його next-об’єкту або, навпаки, про те, що такого перевищення немає.

Управління потоком паралельного алгоритму імітації Петрі-об’єктної моделі



Створення Петрі-об’єктів та зв’язків між ними

```
public static PetriObjModel getModelSMOgroupForTestParallel(int numGroups, int numInGroup) throws ExceptionInvalidNetStructure {
        ArrayList<PetriSim> list = new ArrayList<>();
        int numSMO = numGroups - 1;

        list.add(new PetriSim(libnet.NetLibrary.CreateNetGenerator(2.0)));
        for (int i = 0; i < numSMO; i++) {
            list.add(new PetriSim(libnet.NetLibrary.CreateNetSMOgroup(numInGroup,1, 1.0,"group_"+i))); //   group1,group2,group3...
        }
        list.get(0).getNet().getListP()[1] = list.get(1).getNet().getListP()[0]; //gen = > group1
        list.get(0).addOutT(list.get(0).getNet().getListT()[0]);
        list.get(1).addInT( list.get(1).getNet().getListT()[0]);
        list.get(0).setNextObj(list.get(1));
        list.get(1).setPreviousObj( list.get(0));
        if (numSMO > 1) {
            for (int i = 2; i <= numSMO; i++) {
                int last = list.get(i-1).getNet().getListP().length-1;

                list.get(i).getNet().getListP()[0] = list.get(i-1).getNet().getListP()[last]; //group1 = > group2, group2 = > group3,...

                int lastT = list.get(i-1).getNet().getListT().length-1;
                list.get(i-1).addOutT(list.get(i-1).getNet().getListT()[lastT]);
                list.get(i).addInT(list.get(i).getNet().getListT()[0]);
                list.get(i-1).setNextObj(list.get(i));
                list.get(i).setPreviousObj(list.get(i-1));
        }
    }
   //корегування списку розицій для статистики
        for (int i = 1; i <= numSMO; i++) {
            ArrayList<PetriP> positionForStatistics = new ArrayList<>();
            PetriP[] listP = new PetriP[list.get(i).getNet().getListP().length];
            listP = list.get(i).getNet().getListP();
            for(int j=0;j<listP.length-1;j++){ //окрім останньої, наприклад
                    positionForStatistics.add(listP[j]);
            }
            list.get(i).setListPositionsForStatistica(positionForStatistics);
        }

        PetriObjModel model = new PetriObjModel(list);
        return model;
    }
```

Запуск потоків на виконання

```
PetriObjModel model = getModelSMOgroupForTestParallel(numObj,10);
 model.setTimeMod(time);
 PetriSim.setLimitArrayExtInputs(3);
 model.setIsProtokol(true);

 model.getListObj().forEach((PetriSim e) -> {
     Thread petriObj = new Thread(e);
     threads.add(petriObj);
     petriObj.start();
 });

 threads.forEach((thread) -> {
     try {
         thread.join();
     } catch (InterruptedException ex) {
         Logger.getLogger(TestParallel.class.getName()).log(Level.SEVERE, null, ex);
     }

 });
 printResultsForAllObj(model);
```

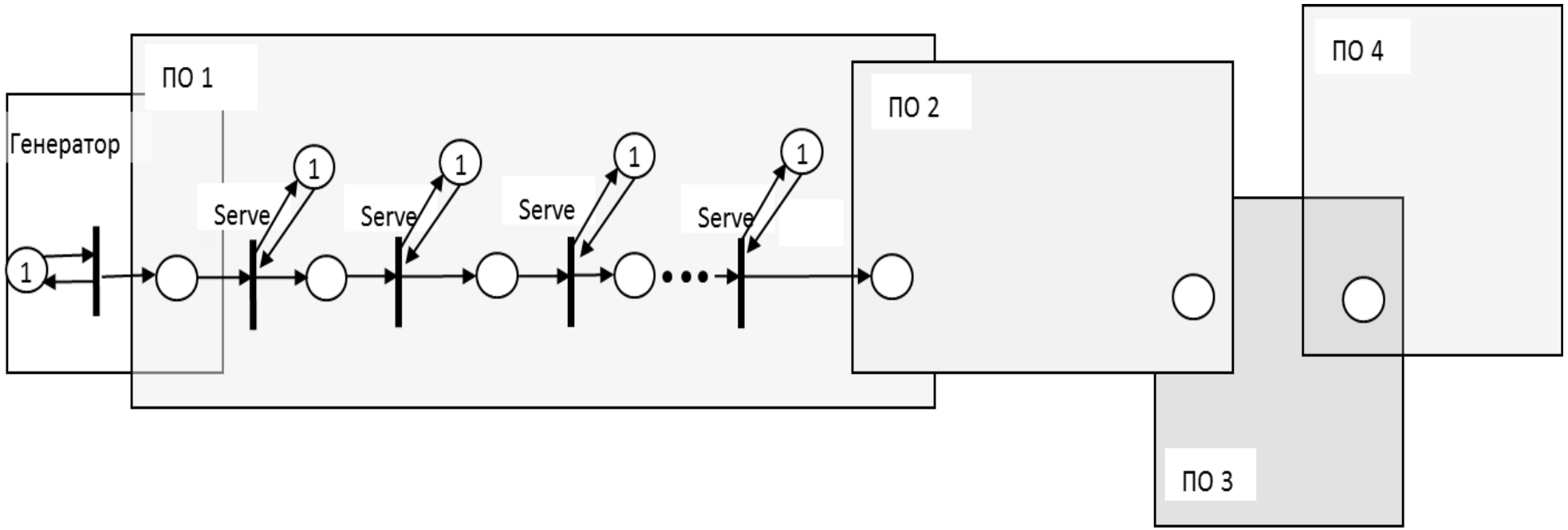
Метод `run()` Петрі-об’єкту

```
@Override
    public void run() {
        while (getTimeLocal() < getTimeMod()) {
            double limitTime = getTimeMod(); //для об єкта без зовнішніх подій
            if (previousObj != null) {
                this.getLock().lock();
                try {
                    while (getTimeExternalInput().isEmpty()) {
                        this.getCond().await();
                    }

                    limitTime = getFirstTimeExternalInput(); //встановлення інтервалу імітації до моменту події входу маркерів в об'єкт ззовні
                    if (limitTime > getTimeMod()) {
                        limitTime = getTimeMod();
                    }
                } catch (InterruptedException ex) {
                    Logger.getLogger(PetriSim.class.getName()).log(Level.SEVERE, null, ex);
                } finally {
                    this.getLock().unlock();
                }
            } else { // previousObj == null
                limitTime = getTimeMod();
            }

            if (getTimeLocal() < limitTime) {
                goUntil(limitTime);
} else { // повідомлення про неправильне визначення limitTime
            return;
            }
        }
}
```

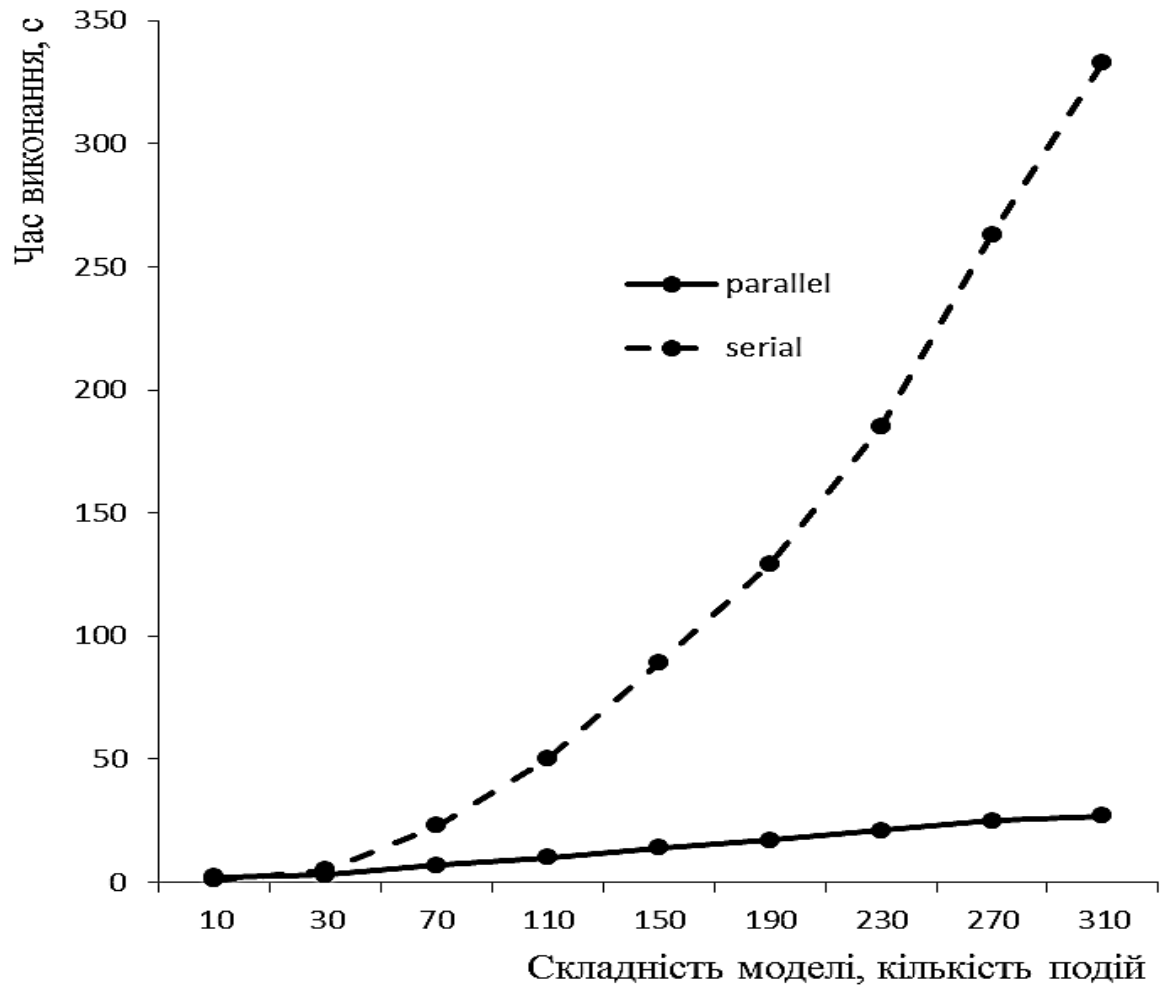
- Повний код програми
  https://github.com/StetsenkoInna/ParallelDESS

Тестування алгоритму

Тестова модель:



Ефективність паралельного алгоритму імітації у випадку локальної змінної часу



Дослідження впливу розміру буферу зовнішніх подій на час виконання алгоритму