Система розподіленого моделювання цифрових логічних схем на обчислювальному кластері

Ладиженський Ю.В., Астахов О.В.
Кафедра прикладної математики та інформатики
Донецький національний технічний університет
ly@cs.dgtu.donetsk.ua, jekvart@gmail.com

Аннотация

Ладыженский Ю.В., Астахов А.В. Система распределенного моделирования цифровых логических схем на вычислительном кластере. *Разработана программная система для распределенного событийно-управляемого логического моделирования на вычислительном кластере Microsoft Windows Compute Cluster Server. Реализованы алгоритмы для экспериментальных исследований параллельного моделирования. Исследована производительность системы в зависимости от протоколов синхронизации и размерности схем. С помощью программной системы достигнуто значительное уменьшение времени моделирования.*

Анотація

Ладиженський Ю.В., Астахов О.В. Система розподіленого моделювання цифрових логічних схем на обчислювальному кластері. *Розроблено програмну систему для розподіленого подійно-керуючого логічного моделювання на обчислювальному кластері Microsoft Windows Compute Cluster Server. Реалізовано алгоритми для експериментальних досліджень паралельного моделювання. Досліджено продуктивність системи залежно від протоколів синхронізації й розмірності схем. За допомогою програмної системи досягнуте значне зменшення часу моделювання.*

Abstract

Ladyzhenskyy Y., Astakhov A. The software system for distributed simulation of digital logical circuits on compute cluster. *The program system for distributed event-driven logic simulation on Microsoft Windows Compute Cluster Server is developed. Algorithms for experimental research of parallel simulations are implemented. The system performance against synchronization protocols and circuit dimension is researched. A considerable time reduction of simulation using a program system is achieved.*

Вступ

Метод імітаційного моделювання широко використовується при проектуванні різних систем. Однак, незважаючи на зростання продуктивності сучасних комп'ютерів, їхньої потужності не вистачає для моделювання задач великої розмірності, коли імітація моделей може тривати годинами. Одним з варіантів вирішення цієї проблеми є застосування паралельного й розподіленого імітаційного моделювання [1]. Для збільшення швидкості обрахунків за допомогою паралельних обчислень та для прискорення процесу моделювання доречно використовувати високопродуктивний обчислювальний кластер.

Кластер – це декілька незалежних обчислювальних машин, що використовуються спільно і працюють як одна система для вирішення тих чи інших задач, наприклад, для підвищення продуктивності, забезпечення надійності, спрощення адміністрування, збільшення швидкості обрахунків за допомогою паралельних обчислень, тощо.

Нещодавно на ринок високопродуктивних обчислень вийшла компанія Microsoft із програмною системою Windows Compute Cluster Server (WCCS). До цього часу основною проблемою в реалізації персональних вирішень для високопродуктивних розрахунків була складність розгортання кластерів і керування ними. WCCS 2003 дозволяє відносно легко здійснювати розгортання обчислювальних вузлів і централізоване керування кластером за допомогою знайомих засобів зі служби каталогів і операційної системи [2, 3, 4].

Незважаючи на велику кількість розподілених систем моделювання, вони працюють не на кластері, або не забезпечують користувача всіма необхідними сервісами для проведення безпосередньо розподіленого моделювання [5]. Майже відсутні системи для розподіленого моделювання цифрових логічних схем. В області реалізації розподілених систем моделювання існує стандарт High Level Architecture (HLA) [6], але багато розробників імітаційних моделей відзначають складність реалізації цього стандарту.

Мета роботи полягає у підвищенні швидкості логічного імітаційного моделювання шляхом організації розподілених обчислювань на кластері. У роботі розглядається аналіз методів розподіленого логічного моделювання, методика побудови і архітектура програмної системи для розподіленого моделювання на обчислювальному кластері. У програмі використовуються методи скорочення витрат пам’яті та прискорення часу за рахунок нової організації структур даних логічних процесів.

Розроблена програмна система дозволяє проводити дослідження розподіленого моделювання, поведінки протоколів синхронізації логічних процесів та значно прискорювати час моделювання цифрових логічних схем у порівнянні з послідовним моделюванням.

Імітаційна модель

Імітаційна модель представляє собою сукупність логічних процесів, які взаємодіють один з одним, посилаючи один одному повідомлення з відміткою часу (time stamped) або події.

Кожний логічний процес виконує роль послідовного симулятора, який підтримує дискретне моделювання. А це означає, що кожний логічний процес містить локальну інформацію про стан об'єктів моделювання й список подій, які були заплановані для цього процесу, але ще не були виконані (pending event list). Цей список неопрацьованих подій включає локальні події (тобто заплановані самим процесом) і події, заплановані для нього іншими процесами [1, 7].

Робота симулятора полягає в тому, щоб вибрати зі списку неопрацьованих подій подію з мінімальною часовою відміткою й обробити її. Так виконання процесу можна розглядати як виконання послідовності подій. Виконання події супроводжується встановленням нових значень змінних величин, які визначають стан об'єкта. Крім того, логічний процес при виконанні чергової події може запланувати виконання нової події для самого себе або для іншого логічного процесу. Кожний логічний процес має локальний годинник. Локальні годинники вказують на час виконання самої останньої обробленої симулятором події. Час, на який заплановано логічним процесом будь-яка подія, повинний бути більше, ніж значення його локального годинника.

Події логічного процесу повинні виконуватися в хронологічному порядку. Цю вимогу називають обмеженням локальної каузальності (local causality constraint). Ця властивість дозволяє переконатися в тому, що виконання імітаційного прогону повторюється (дає ті ж результати при тому самому наборі вихідних даних) [8].

Розподілену модель, що складається з N логічних процесів, можна визначити у такий спосіб [1]:

$$
\begin{aligned}
&DM = \bigcup_{k=1}^{N} LP_k \\
&LP_k = \{sm_k, T_k, S_k\}, \quad T_k = t_{1,k} \\
&S_k = \{(Q_{1,k}, t_{1,k}), \ldots, (Q_{m,k}, t_{m,k})\},
\end{aligned}
\tag{1}
$$

де DM – розподілена імітаційна модель, LP – логічний процес, sm – модель процесу, T – локальний час моделювання, S – локальний список подій, Q – подія або повідомлення, t – модельний час події Q.

Метою механізму синхронізації модельного часу є виконання кожним логічним процесом подій у порядку зростання їхніх відміток часу.

Це вимога відома як локальне обмеження причинного зв'язку, тому що вона забезпечує імітацію природного порядку від причини до наслідку.

Алгоритми синхронізації модельного часу поділяються на два основних класи: консервативні й оптимістичні. З більш докладною класифікацією алгоритмів синхронізації модельного часу можна ознайомитися в [7, 8, 9].

Перші алгоритми синхронізації використовували консервативний підхід. Принципове завдання консервативного протоколу – визначити час, коли обробка чергової події зі списку неопрацьованих подій є “безпечною”. Іншими словами, подія є безпечною, якщо можна гарантувати, що процес надалі не одержить від інших процесів події з меншою відміткою часу [7].

На відміну від консервативних алгоритмів, що не допускають порушення обмеження локальної каузальності, оптимістичні методи не стежать за цим обмеженням. Однак цей підхід гарантує виявлення порушення каузальності і його усунення. Оптимістичний метод має дві важливих переваги в порівнянні з консервативним. По-перше, йому властивий більш високий рівень паралелізму. Дійсно, якщо дві події можуть впливати один на одного, але алгоритм такий, що насправді цього немає, то оптимістичний програмний механізм дозволяє обробляти події паралельно на відміну від консервативного. Консервативний у цьому випадку однаково вимагає послідовного виконання цих двох подій. По-друге, консервативний механізм звичайно вимагає додаткової інформації, наприклад, відстань між об'єктами. Це необхідно для визначення безпечної події процесу. Оптимістичні алгоритми, що використовують таку інформацію, теж виконуються швидше, але вплив цієї інформації на коректність виконання набагато менше. Оптимістичний алгоритм, таким чином, більш прозорий при розробці математичного програмного забезпечення, чим консервативний, розробка програмного забезпечення спрощується. З іншого боку, для оптимістичного підходу можуть знадобитися додаткові обчислення, що знижує ефективність його застосування [10].

Архітектура програмної системи

Програмна система для моделювання цифрових логічних схем реалізована в середовищі проектування Microsoft Visual Studio 2005 на мові C++ і працює під керуванням операційної системи сімейства Windows на обчислювальному кластері Windows Compute Cluster Server 2003. Система спроектована за допомогою об'єктно-орієнтованого підходу. Передача повідомлень між вузлами кластера реалізована за допомогою протоколу MPI (Message Passing Interface) [11, 12]. Система містить у собі 4 файли з реалізаціями алгоритмів синхронізації.

Система дозволяє проводити послідовне моделювання. Для синхронізації процесів при розподіленому моделюванні реалізований консервативний протокол з маркером і з нульовими повідомленнями, оптимістичний протокол без глобального віртуального часу та з глобальним віртуальним часом. Для зручності роботи із програмною системою розроблений програмний інтерфейс на мові C#, що дозволяє створювати й запускати завдання на кластері, здійснювати моніторинг виконання завдання й аналізувати результати моделювання [13].

Програмне середовище для розподіленого моделювання цифрових логічних схем і дослідження протоколів синхронізації логічних процесів містить у собі підсистему введення логічних схем і вхідного впливу, підсистему розподіленого логічного моделювання й підсистему аналізу результатів моделювання.

Архітектура програмного середовища представлена на рис. 1.

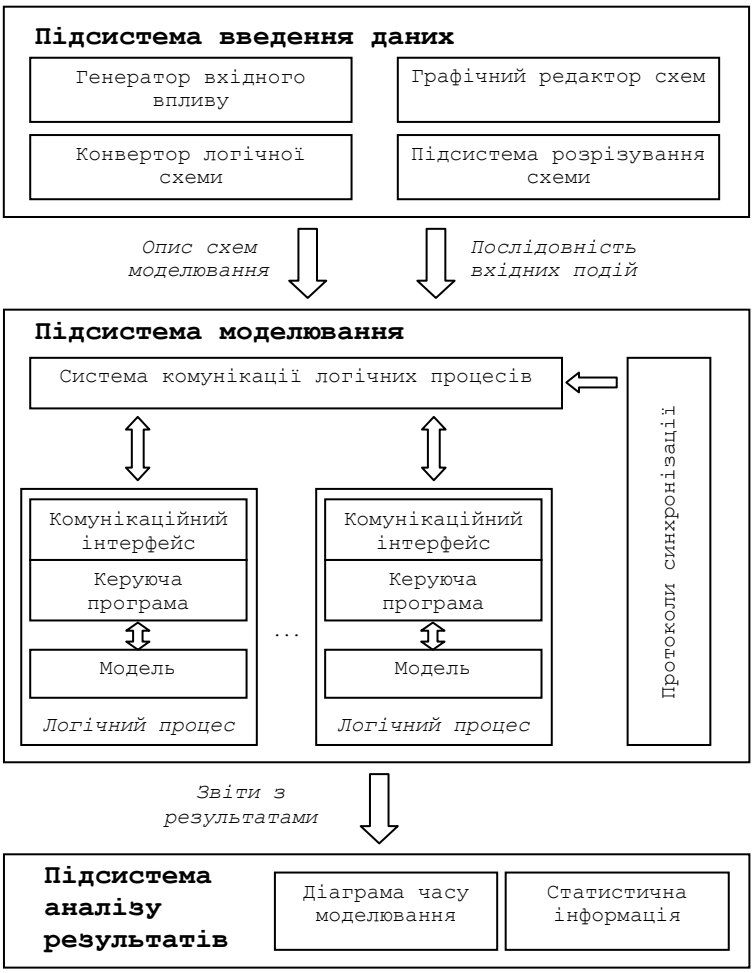


Рисунок 1 – Архітектура програмної системи

Прикладні програми, що входять у підсистему введення даних, і програма для побудови діаграм часу моделювання є зовнішніми. Їх

розробив у середовищі проектування Delphi на той час аспірант Донецького національного технічного університету Попов Ю.В [14, 15].

Моделюючий логічний процес

Основним компонентом розподіленої імітаційної системи є моделюючий логічний процес. Він відповідає за моделювання однієї частини логічної схеми.

Кожний логічний процес завантажує послідовність вхідних даних у локальний список подій і опис частини схеми моделювання. При цьому завантажується інформація про функціональні елементи й відповідні їм таблиці істинності з файлів, що знаходяться у бібліотеці елементів, шлях до якої прописується серед атрибутів проекту моделювання. За функціональним елементом цифрової логічної схеми закріплюється тільки посилання на таблицю істинності, у такий спосіб кількість завантажених таблиць істинності може бути значно менше, ніж загальне число функціональних елементів у логічній схемі.

На підставі отриманої інформації про структуру схеми формується карта глобальних входів і карта глобальних виходів, що містить відображення глобального імені вузла на його локальне ім'я в схемі, а також створюється карта внутрішніх входів-виходів, що дозволяє обчислити по імені вузла номер функціонального елемента, з яким зв'язаний даний вузол, і номер цього виходу на елементі. Ці структури дозволяють швидко одержати доступ до необхідного елемента в схемі та встановити нове значення сигналу на вході або виході.

Зв'язок комунікаційної системи з логічним процесом здійснюється через його комунікаційний інтерфейс. Комунікаційний інтерфейс призначений для керування глобальним порядком обробки повідомлень. Для цього в комунікаційному інтерфейсі є вхідні й вихідні буфери. У вхідних буферах зберігаються події, які приходять із інших процесів, у вихідних – події, які необхідно відправити. Для кожного вхідного буфера в комунікаційному інтерфейсі є канальні годинники, які показують локальний час на процесі. Канальні годинники містять копію часу першого повідомлення у вхідному буфері. Оскільки події приходять строго в хронологічному порядку, цей час відповідає часу останнього повідомлення, що прийшло по цьому каналу.

Просуванням локального модельного часу на логічному процесі займається координатор процесу моделювання. Цей координатор також зберігає горизонт часу моделювання для консервативних алгоритмів синхронізації та обчислює його щоразу після прийому всіх подій із вхідних буферів.

Для реалізації списку локальних подій, черг, карт відображення вузлів схеми використовуються стандартні контейнери бібліотеки STL (Standard Template Library).

Архітектура консервативного логічного процесу моделювання представлена на рис. 2.

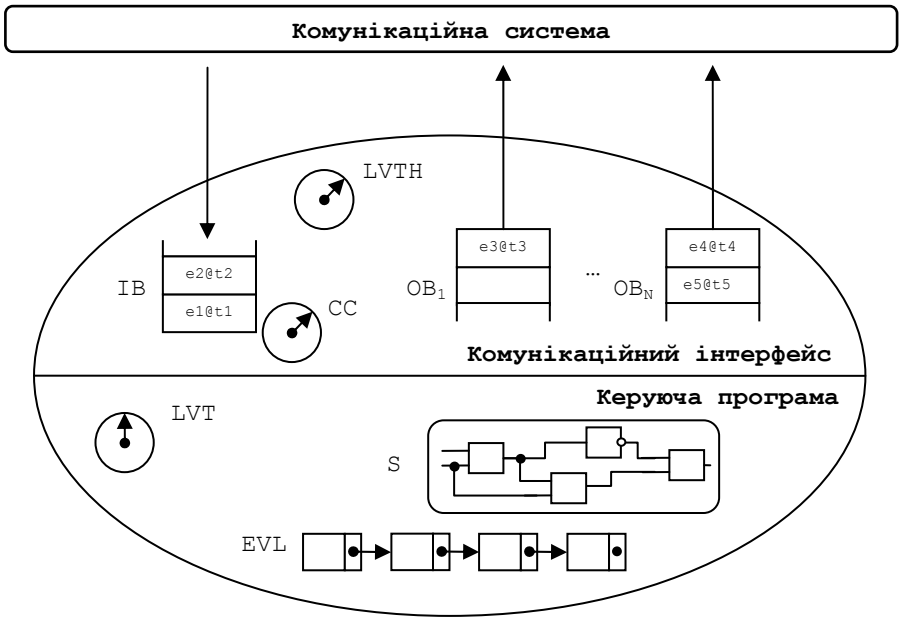


Рисунок 2 – Архітектура консервативного логічного процесу

Для кожного із протоколів моделювання логічний процес містить локальні віртуальні годинники (LVT), схему моделювання (S) та локальний список подій (EVL).

Архітектура консервативного логічного процесу відрізняється від класичної схеми, яку запропонував Ferscha [7]. Відмінність складається в наявності єдиного вхідного буфера. З тієї причини, що повинні бути оброблені всі вхідні події від процесів-відправників, необхідність в окремих буферах для кожного логічного процесу відпадає. Інформація ж про те, з якого логічного процесу прийшло повідомлення міститься в назві вузла зовнішньої події.

При такій структурі кількість канальних годинників (CC) буде дорівнювати одному, і вони будуть показувати той же самий час моделювання, що й горизонт локального віртуального часу (LVTH).

Число буферів, як і в класичній схемі, стільки ж, скільки логічних процесів, які мають вхідну лінію зв'язку з даного моделюючого процесу.

Оптимістичний логічний процес містить додаткові структури даних для зберігання стану системи в минулому (SS) та інформацію про відправлені повідомлення (OQ). Архітектура оптимістичного логічного процесу представлена на рис. 3.

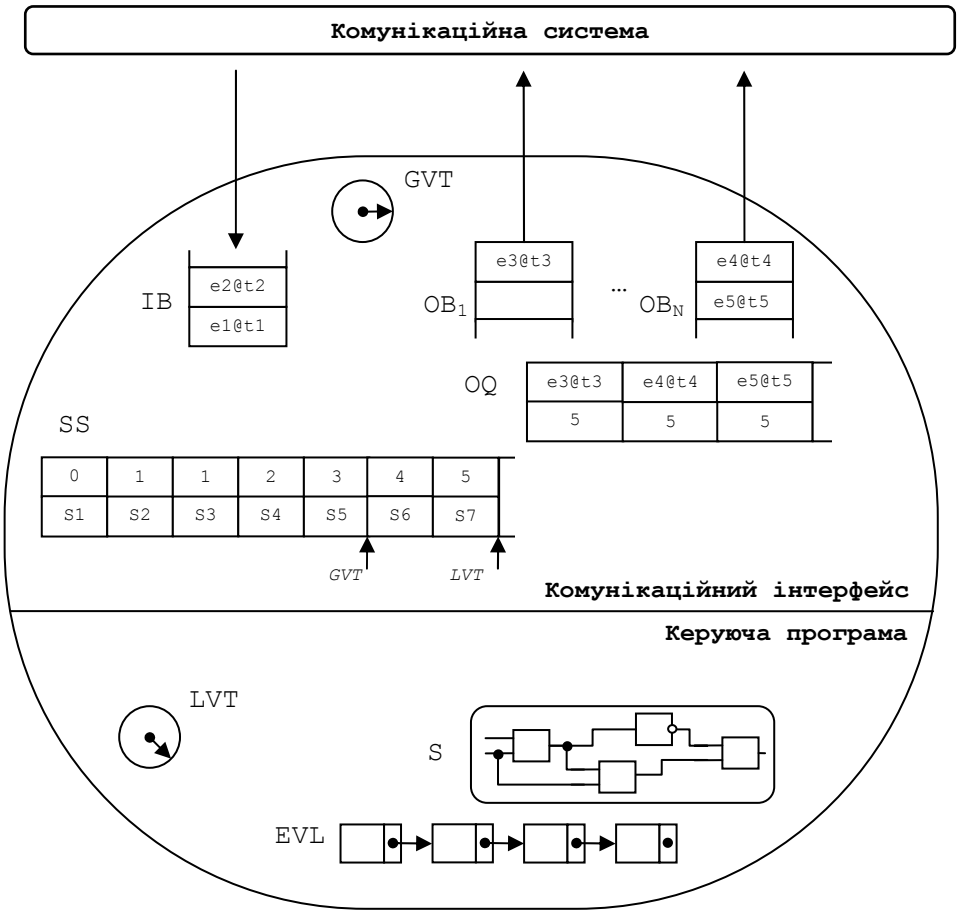


Рисунок 3 – Архітектура оптимістичного логічного процесу

Дана реалізація логічного процесу не містить буфера пам'яті для зберігання вхідних повідомлень, тому що цю роль бере на себе звичайний вхідний буфер. Горизонт локального віртуального часу заміняється на глобальний віртуальний час (GVT), що дозволяє очищати пам'ять у структурах даних до позначки часу зі значенням GVT.

У буфер SS записується контрольна точка поточного стану й час запису. Щоразу із появою нової події черги проглядаються на наявність цієї події в черзі. Якщо дана подія буде знайдена в якій-небудь із черг, це буде говорити про те, що прийшло антиповідомлення, після чого подію-дублікат потрібно видалити.

Система комунікації процесів

Взаємодію між логічними процесами організовано через комунікаційну систему, що представлена в програмній системі як окремий компонент. Система комунікації реалізує логіку роботи алгоритму синхронізації моделюючих процесів. Передача повідомлень реалізована за допомогою протоколу MPI (Message Passing Interface).

Увесь обмін даних у MPI здійснюється в рамках комунікаторів, які визначають контекст обміну й групу процесів, які з ними зв'язані. Визначений комунікатор MPI_COMM_WORLD представляє єдиний контекст і сукупність всіх MPI процесів, що виконуються. Всі логічні процеси, що беруть участь у моделюванні належать даному комунікатору.

В алгоритмах синхронізації використовуються асинхронні посилки (MPI_Send) і прийоми (MPI_Recv). Стандартна передача даних за допомогою функції MPI_Send вважається завершеною, як тільки повідомлення відправлене, незалежно від того, чи буде воно отримане іншим логічним процесом чи ні. Передача повідомлення може починатися навіть якщо не почався його прийом [12].

Консервативний протокол з нульовими повідомленнями

Консервативний алгоритм синхронізації з нульовими повідомленнями для запобігання виникнення “парадоксів часу” використовує відправлення порожніх подій (нульових повідомлень) логічним процесам, у яких є вхідна лінія зв'язку від даного процесу [16, 17].

Нульові повідомлення відсилаються по вихідній лінії зв'язку після обробки чергової події в тому випадку, якщо значення сигналу на виході залишилося незмінним (рис. 4).

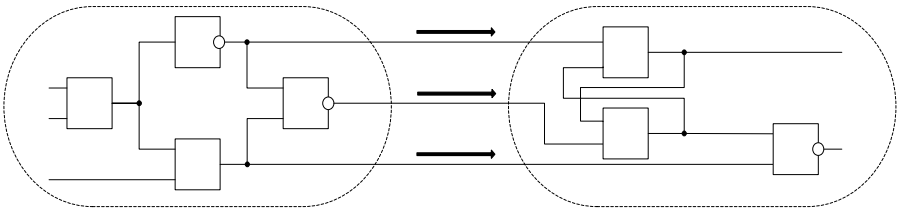


Рисунок 4 – Посилка нульових повідомлень

Таким чином, максимальна кількість нульових повідомлень може дорівнювати числу вихідних зв'язків логічного процесу.

Алгоритм протоколу з нульовими повідомленнями приведений на рис. 5.

Повідомлення, які передаються через комп'ютерну мережу, об'єднуються в один загальний масив. Для того щоб на початку моделювання логічні процеси не очікували прийому подій один від одного виконується перша посилка нульових повідомлень по всіх вихідних лініях зв'язку процесу (S1).

```
S1          for all LP[i] do
                if ( outLink[i] ) then Send_to_LP[i](null_message); end if;
            od for;

S2          while ( EVL.notEmpty or LVTH <> MAX_LVTH ) do

S2.1                for all LP[i] do
                        if ( inLink[i] ) then
                                Recv_from_LP[i](new_message);
                                if ( new_message == done_message ) then
                                        outLink[i] = inLink[i] = false;
                                else
                                        inBuffer.add(new_message);
                                end if;
                        end if;
                od for;

S2.2                LVTH.calculate();
S2.3                inBuffer.copy_to(EVL);

S2.4                LVT = EVL.vt;
S2.5                if ( LVT >= FinishVT ) break; end if;

S2.6                while ( EVL.vt == LVT ) do
                        Set_input(EVL.node, EVL.value);
                        EVL.delete();
                od while;

S2.7                for all outputs[i] do
                        calculate_all_outputs();
S2.7.1                  if (out.global) then
                          if (new_value <> old_value ) then
                            out.value = new_value;
                            new_message = (out.name, out.value, LVT + delay);
                            outBuffer[out.proc].add(new_message);
                          else
                            outBuffer[out.proc].add(null_message);
                          end if;
S2.7.2                  else
                          if (new_value <> old_value ) then
                            out.value = new_value;
                            new_message = (out.name, out.value, LVT + delay);
                            EVL.add(new_message);
                          end if;
                        end if;
                end for;

S2.8                for all LP[i] do
                      if ( outLink[i] ) then Send_to_LP[i](outBuffer[i]); end if;
                od for;

            end while;

S3          for all LP[i] do
                if ( outLink[i] ) then Send_to_LP[i](done_message); end if;
            od for;
```

Рисунок 5 – Алгоритм консервативного протоколу з нульовими повідомленнями

У циклі, поки список локальних подій не буде порожнім або існують події від інших процесів, виконується моделювання схеми (S2). Спочатку здійснюється прийом повідомлень від інших процесів (S2.1), після чого обчислюється горизонт часу (S2.2) й нові повідомлення розміщуються у локальному списку подій (S2.3). Поточний локальний віртуальний час приймається за значення позначки часу в голові локального списку подій

(S2.4). При досягненні кінцевого віртуального часу моделювання процес завершується навіть у тому випадку, якщо локальний список подій не порожній (S2.5). Потім обробляються всі події з поточною відміткою часу (S2.6). Для кожного виходу функціонального елемента перевіряється значення сигналу (S2.7). Якщо встановилося нове значення сигналу на виході, то відсилається відповідна подія. При цьому, може генеруватися внутрішня подія (S2.7.2) або зовнішня (S2.7.1), яку потрібно відправити іншому логічному процесу (S2.8).

Інформація про зміну глобального виходу на схемі фіксується й записується у файл звіту моделювання. При завершенні моделювання логічний процес здійснює посилку так званої “завершальної події” (S3), при одержанні якої процес більше не буде переглядати вхідні лінії зв'язку з процесу-відправника.

Консервативний протокол з маркерами

Консервативний алгоритм із маркерами при відправленні повідомлень іншому логічному процесу використовує значення “попереднього перегляду” [16]. Маркер представляє собою повідомлення, відмітка модельного часу якого дорівнює мінімальному значенню віртуального часу серед всіх повідомлень, які можуть бути відіслані. При цьому мінімальна кількість повідомлень, які можуть бути відіслані іншому процесу, дорівнює одному (рис. 6). Для того, щоб процес-одержувач знав скільки необхідно прийняти нових повідомлень, процес-відправник посилає повідомлення про кількість подій.

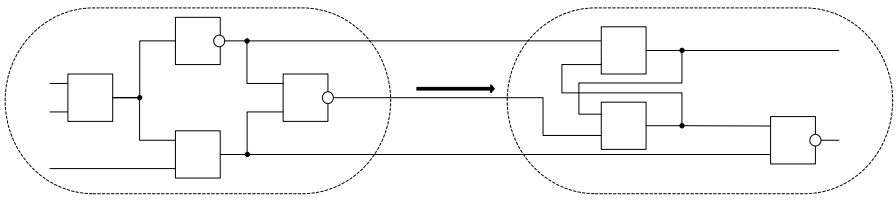


Рисунок 6 – Посилка маркер-повідомлення

Оптимістичний протокол без глобального часу

Якщо консервативні алгоритми виключають навіть потенційну можливість виникнення парадоксу часу при моделюванні, то оптимістичні алгоритми “сподіваються”, що при паралельному виконанні логічних процесів потенційна можливість виникнення парадоксу часу не стане реальністю [7].

Алгоритм оптимістичного протоколу синхронізації логічних процесів приведений на рис. 7.

```
S1          for all LP[i] do
                  if ( outLink[i] ) then Send_to_LP[i](0); end if;
            od for;

S2          while ( not allDone ) do
                   for all LP[i] do
                          if ( inLink[i] ) then
S2.1                         Recv_from_LP[i](count);
S2.2                         for j=0 to count do
                                Recv_from_LP[i](new_message);
                             od for;
S2.3                         if ( new_message.vt >= LVT ) then
                                    if ( new_message in inBuffer ) then
                                           inBuffer.delete(new_message);
                                           EVL.delete(new_message);
                                    else
                                           inBuffer.add(new_message);
                                           EVL.add(new_message);
                                    end if;
S2.4                             else
S2.4.1                              if ( new_message in inBuffer ) then
                                           inBuffer.delete(new_message);
                                    else
                                           inBuffer.add(new_message);
                                    end if;

S2.4.2                              rollback();
S2.4.3                              for all LP[i] do
                                    if ( outLink[i] ) then
                                       antimessage = outBufMemeory[i].msg;
                                       Send_to_LP[i](antimessage);
                                       outBufMemeory[i].delete();
                                    end if;
                                od for;
                             end if;
                          end if;
                   od for;

S2.5               LVT = EVL.vt;
S2.6               if ( LVT >= FinishVT ) done = true; else done = false; end if;
S2.7               while ( EVL.vt == LVT ) do
                          Set_input(EVL.node, EVL.value);
                          EVL.delete(); Save_state();
                   od while;

S2.8               for all outputs[i] do
                          calculate_all_outputs();
S2.8.1                    if (out.global) then
                               if (new_value <> old_value ) then
                                  out.value = new_value;
                                  new_message = (out.name, out.value, LVT + delay);
                                  outBuffer[out.proc].add(new_message);
                                  Save_state();
                               end if;
S2.8.2                    else
                               if (new_value <> old_value ) then
                                  out.value = new_value;
                                  new_message = (out.name, out.value, LVT + delay);
                                  EVL.add(new_message); Save_state();
                               end if;
                          end if;
                   end for;

S2.9               for all LP[i] do
                          if ( outLink[i] ) then
                                  outBuffer[i].copy_to(outBufMemory[i]);
                                  Send_to_LP[i](outBuffer[i].length);
                                  Send_to_LP[i](outBuffer[i]);
                           end if;
                   od for;

S2.10              for all LP[i] do Send_to_LP[i](done); od for;

            end while;
```

Рисунок 7 – Алгоритм оптимістичного протоколу

У випадку ж виникнення парадокса часу оптимістичні алгоритми реалізують “відкіт” (rollback) логічного процесу до значення модельного часу, у який йому було відіслане повідомлення, що викликало парадокс часу. Відкіт містить у собі ліквідацію наслідків некоректного виконання логічного процесу й повторне виконання цього процесу з урахуванням повідомлення, що викликало парадокс часу.

Будь-яке повідомлення, що приходить на моделюючий процес зберігається у буфері вхідних повідомлень. Стан схеми фіксується контрольними точками. Зміна якого-небудь входу або виходу в схемі записується в буфер станів, при цьому відмічається локальний віртуальний час моделювання, під час якого відбувся запис. Повідомлення, які відправляються з логічного процесу, зберігаються у відповідному вихідному буфері.

У той момент, коли приходить подія з минулого (з відміткою часу меншою, чим значення локального годинника), відбувається відкіт логічного процесу. Стан схеми відновлюється на момент до появи події з минулого, оброблені вхідні повідомлення знову записуються в локальний список подій. Події, які зберігаються в буфері відправлених повідомлень і були відіслані після появи події, що викликала відкіт, відправляються повторно. Ці повідомлення будуть у ролі антиповідомлень для іншого логічного процесу. щоразу, коли приходить нове повідомлення, перевіряться наявність цього повідомлення у вхідному буфері. Якщо це повідомлення вже присутнє там, то необхідно його видалити з буфера й з локального списку подій, якщо воно вже туди було записано.

Як і у випадку з консервативним протоколом моделювання, в оптимістичному протоколі перед початком моделювання логічні процеси роблять першу посилку процесам, які мають лінії зв'язку з ними, для того, щоб не заблокувати процеси (S1). Це може відбутися в схемі подібній до тригера, де кожний з логічних процесів має зворотний зв'язок з іншого процесу. У цьому випадку процеси будуть очікувати подій один від одного й моделювання не зможе розпочатися.

Цикл моделювання виконується до тих пір, доки хоча б один логічний процес не завершив моделювання (S2). Кожний логічний процес спочатку здійснює прийом значення кількості майбутніх повідомлень (S2.1), а потім – самі повідомлення (S2.2). Якщо відмітка часу нового повідомлення більше поточного часу моделювання (S2.3), то перевіряється вхідний та локальний буфер на наявність цього повідомлення та додається до них при його відсутності. Якщо повідомлення прийшло “з минулого” (S2.4), то треба перевірити чи не є воно антиповідомленням (S2.4.1), зробити відкіт (S2.4.2) та надіслати нові антиповідомлення (S2.4.3). Поточний локальний віртуальний час приймається за значення позначки часу в голові локального списку подій (S2.5). Моделювання завершується, коли кожний логічний процес повідомив про своє завершення (S2.6).

Потім обробляються всі події з поточною відміткою часу (S2.7). Для кожного виходу функціонального елемента перевіряється значення сигналу (S2.8). Якщо встановилося нове значення сигналу на виході, то відсилається відповідна подія: внутрішня (S2.8.1) або зовнішня (S2.8.2), яку потрібно відправити іншому логічному процесу (S2.9). Процеси, які закінчили свою роботу, очікують можливих нових повідомлень, які можуть викликати відкіт. Повідомлення зі станом про готовність процесу моделювання відсилається всім процесам, що беруть участь у моделюванні, перед черговим прийомом повідомлень із інших процесів (S2.10).

Дана реалізація оптимістичного алгоритму не стежить за звільненням пам'яті в структурах даних, які зберігають інформацію про стан схеми в контрольних точках, а також про прийняті й відправлені повідомлення, до завершення моделювання.

Оптимістичний протокол з глобальним часом

Інша реалізація оптимістичного алгоритму передбачає оптимізацію його роботи завдяки обчисленню нижньої межі відміток часу будь-якого майбутнього відкоту GVT (Global Virtual Time або глобальний віртуальний час моделювання) [10].

Для обчислення GVT використовується техніка, аналогічна тієї, яка використовується для обчислення горизонту часу в консервативних алгоритмах. Якщо обчислено поточне значення GVT, то пам'ять, яка використовується для зберігання станів всіх логічних процесів (контрольні точки) з відмітками часу меншими, чим GVT, може бути очищена. Значення GVT у розподілених моделях, що використовують оптимістичні алгоритми синхронізації, є аналогом глобального модельного часу в послідовних моделях.

Алгоритм оптимістичного протоколу із глобальним віртуальним часом у порівнянні з неоптимізованою версією містить додатковий програмний код, у якому з кожного логічного процесу розсилається поточне значення локального віртуального часу. Мінімальний час серед всіх показань локальних годинників для кожного процесу і буде глобальним віртуальним часом.

Пам'ять, що займається у вхідному й вихідному буфері повідомлень, очищається до відмітки часу, яка дорівнює GVT. Пам'ять у буфері станів схеми при цьому не очищається, тому що інформація, що міститься в ньому, використовується для формування звіту процесу моделювання.

Дана реалізація алгоритму дозволяє економити не тільки ресурси пам'яті, але й скоротити час на пошук у структурах даних, тому що при тривалому часі моделювання або ж великій кількості вхідних даних довжина цих структур може бути дуже великою.

Запуск процесу моделювання

Перед початком моделювання потрібно підготувати всі необхідні вхідні дані. Повинен бути сформований файл схеми моделювання з описом схеми відповідного формату та створений файл із послідовністю вхідних подій для кожного глобального вузла цифрової логічної схеми. Для розподіленого моделювання схема розрізається на задане число частин за допомогою спеціальної прикладної програми. Правильність розрізування й карту зв'язків для даної схеми можна переглянути у відповідному файлі, що генерується перед початком моделювання. При необхідності створюється файл проекту, у якому вказуються значення параметрів процесу моделювання.

Результатами роботи розподіленої системи моделювання цифрових логічних схем є звіти, у яких фіксуються зміни сигналів на виходах схеми, а також статистична інформація про хід моделювання.

Для зручності роботи із програмною системою була розроблена програмна оболонка, що дозволяє швидко формувати й запускати завдання на кластері, здійснювати моніторинг виконання завдання й аналізувати результати моделювання [13].

На вкладці "Create Task" (рис. 8) створюється нова задача.

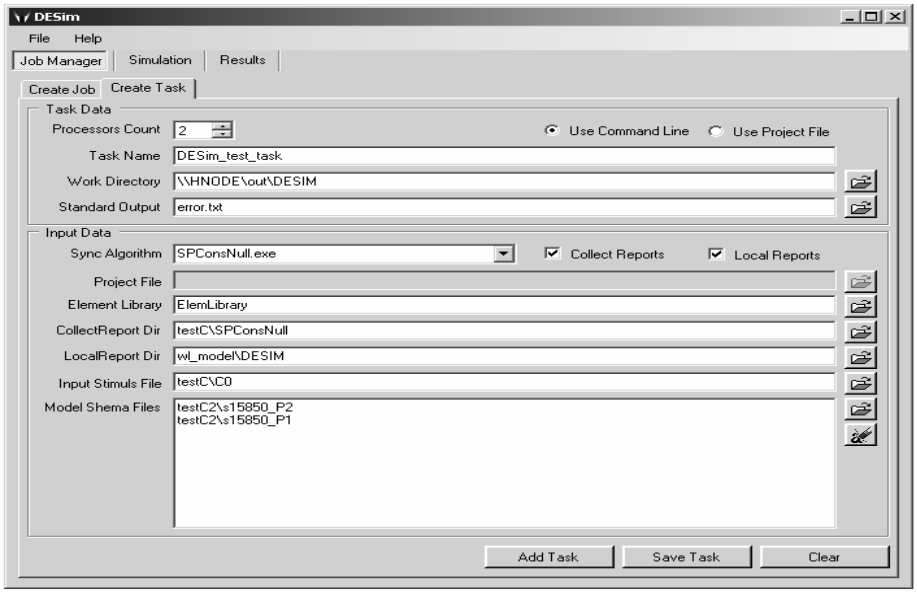
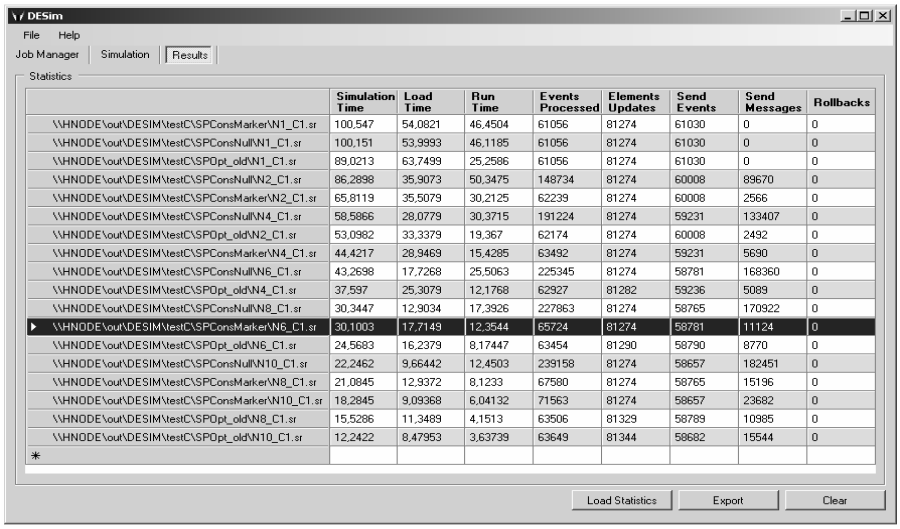


Рисунок 8 – Редагування нової задачі у програмному інтерфейсі

Для аналізу результатів моделювання в програмному інтерфейсі на вкладці "Results" можна завантажити статистичну інформацію зі звітів моделювання або з файлів статистики в таблицю (рис. 9).



	Simulation Time	Load Time	Run Time	Events Processed	Elements Updates	Send Events	Send Messages	Rollbacks
\\HNODE\out\DESIM\testC\SPConsMarker\N1_C1.sr	100,547	54,0821	46,4504	61056	81274	61030	0	0
\\HNODE\out\DESIM\testC\SPConsNull\N1_C1.sr	100,151	53,9993	46,1185	61056	81274	61030	0	0
\\HNODE\out\DESIM\testC\SPOpt_old\N1_C1.sr	89,0213	63,7499	25,2586	61056	81274	61030	0	0
\\HNODE\out\DESIM\testC\SPConsNull\N2_C1.sr	86,2898	35,9073	50,3475	148734	81274	60008	89670	0
\\HNODE\out\DESIM\testC\SPConsMarker\N2_C1.sr	65,8119	35,5079	30,2125	62239	81274	60008	2566	0
\\HNODE\out\DESIM\testC\SPConsNull\N4_C1.sr	58,5866	28,0779	30,3715	191224	81274	59231	133407	0
\\HNODE\out\DESIM\testC\SPOpt_old\N2_C1.sr	53,0982	33,3379	19,367	62174	81274	60008	2492	0
\\HNODE\out\DESIM\testC\SPConsMarker\N4_C1.sr	44,4217	28,9469	15,4285	63492	81274	59231	5690	0
\\HNODE\out\DESIM\testC\SPConsNull\N6_C1.sr	43,2698	17,7268	25,5063	225345	81274	58781	168360	0
\\HNODE\out\DESIM\testC\SPOpt_old\N4_C1.sr	37,597	25,3079	12,1768	62927	81282	59236	5089	0
\\HNODE\out\DESIM\testC\SPConsNull\N8_C1.sr	30,3447	12,9034	17,3926	227863	81274	58765	170922	0
\\HNODE\out\DESIM\testC\SPConsMarker\N6_C1.sr	30,1003	17,7149	12,3544	65724	81274	58781	11124	0
\\HNODE\out\DESIM\testC\SPOpt_old\N6_C1.sr	24,5683	16,2379	8,17447	63454	81290	58790	8770	0
\\HNODE\out\DESIM\testC\SPConsNull\N10_C1.sr	22,2462	9,66442	12,4503	239158	81274	58657	182451	0
\\HNODE\out\DESIM\testC\SPConsMarker\N8_C1.sr	21,0845	12,9372	8,1233	67580	81274	58765	15196	0
\\HNODE\out\DESIM\testC\SPConsMarker\N10_C1.sr	18,2845	9,09368	6,04132	71563	81274	58657	23682	0
\\HNODE\out\DESIM\testC\SPOpt_old\N8_C1.sr	15,5286	11,3489	4,1513	63506	81329	58789	10985	0
\\HNODE\out\DESIM\testC\SPOpt_old\N10_C1.sr	12,2422	8,47953	3,63739	63649	81344	58682	15544	0

Рисунок 9 – Аналіз результатів моделювання у програмному інтерфейсі

Експериментальні дослідження алгоритмів синхронізації

Проведення експериментів було здійснено на схемах, створених за допомогою графічного редактора логічних схем, і на схемах з міжнародного каталогу ISCAS-89 [18].

Таблиця 1. Тестові схеми моделювання

Назва схеми	Входів / виходів	Число тригерів	Число інверторів / Число функц. елементів
s15850	14/87	597	6324/3448
s5378	35/49	179	1775/1004
s1196	14/14	18	141/388
s38584	12/278	1452	7805/11448

Кожна з цифрових логічних схем була промодельована на 1, 2, 4, 6, 8 і 10 процесорах.

Всі послідовності вхідних подій генерувалися за допомогою програми STIGenerator [14]. Затримка появлення чергового повідомлення становить від 3 до 10 тактів модельного часу. Порівняння алгоритмів синхронізації проводилося на однакових логічних схемах з однаковими тестовими даними.

За результатами моделювання (рис. 10) видно, що швидше працює оптимістичний алгоритм із глобальним віртуальним часом і повільніше – консервативний з нульовими повідомленнями. Час моделювання зі збільшенням числа процесів за допомогою кожного із протоколів моделювання значно зменшується. При послідовному моделюванні консервативний і оптимістичний протоколи дають різні показники часу. Це пов'язане зі спрощеною реалізацією оптимістичного алгоритму.

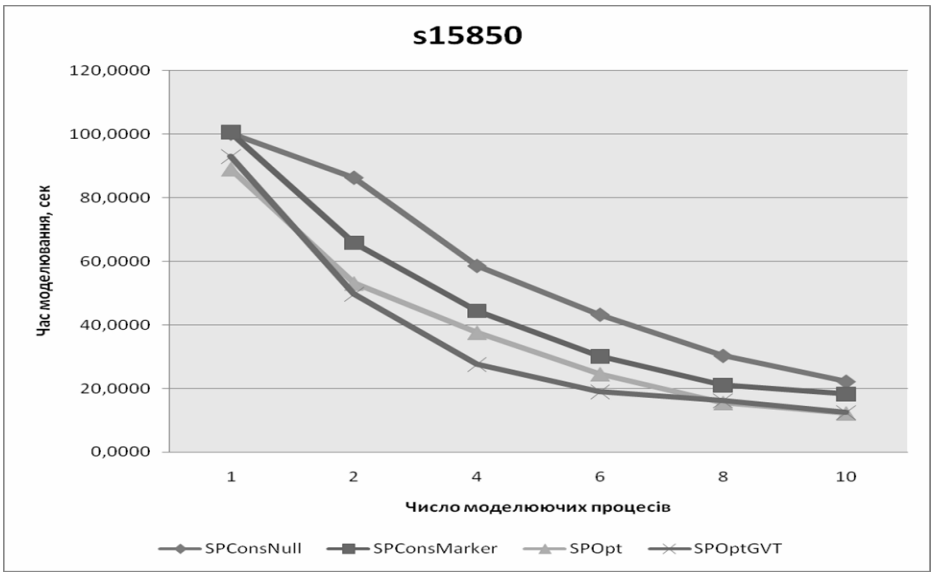


Рисунок 10 – Залежність часу моделювання від числа процесів на схемі s15850

Однак отримані значення багато в чому залежать від схеми моделювання, якості розрізування схеми, послідовності вхідних подій, загального часу моделювання й можливих затримок у комп'ютерній мережі. Так на графіку для однієї зі схем (рис. 11) видно, що на 8 процесорах схема була розрізана невдало.

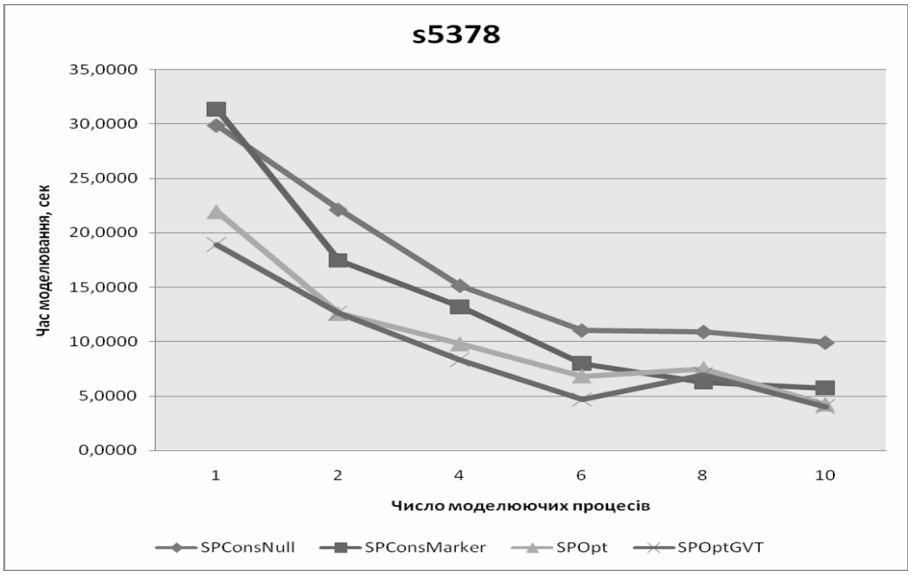


Рисунок 11 – Залежність часу моделювання від числа процесів на схемі 5378

Консервативні алгоритми при цьому не дали значного прискорення у часі, а за допомогою оптимістичних алгоритмів був отриманий результат гірший, ніж на попередньому кроці. Для обох наведених схем видно, що на 8 і 10 процесах оптимістичні алгоритми дають однакові результати.

Збільшення часу моделювання й числа вхідних даних навіть для порівняно невеликої схеми (s1196) негативно позначається на моделюванні за допомогою оптимістичного алгоритму синхронізації, що не звільняє пам'ять (рис. 12).

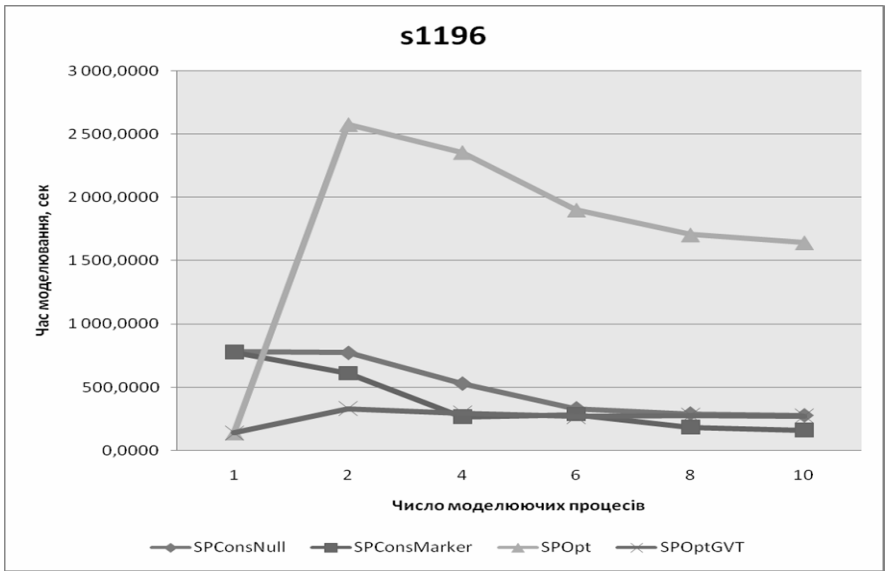


Рисунок 12 – Залежність часу моделювання від числа процесів на схемі s1196

Час моделювання на 2 процесах за допомогою оптимістичного алгоритму без глобального віртуального часу збільшується майже в 18 разів у порівнянні з послідовним моделюванням.

Для малого ж числа вхідних даних навіть на великих за розміром схемах цей алгоритм може працювати швидше оптимізованого оптимістичного алгоритму (рис. 13).

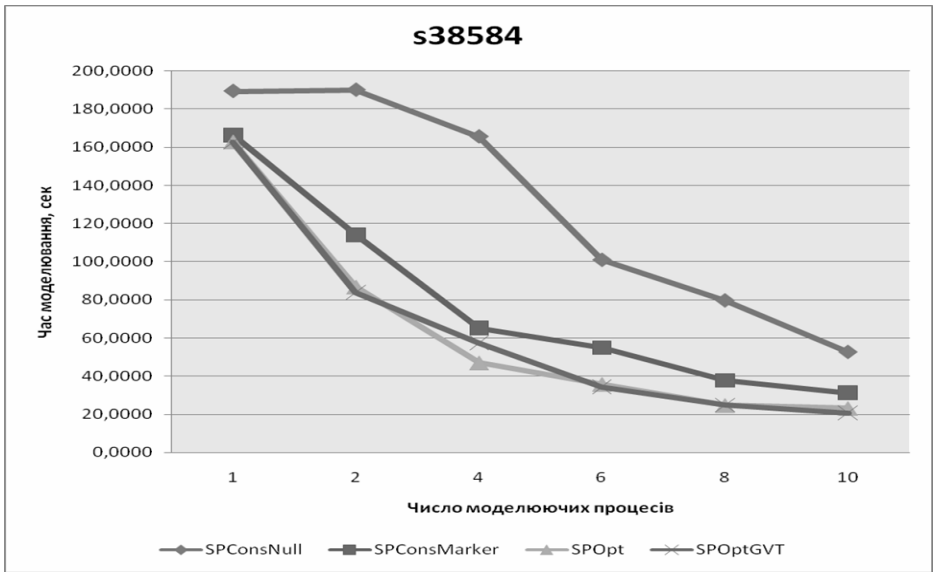


Рисунок 13 – Залежність часу моделювання від числа процесів на схемі s38584

Консервативний алгоритм із нульовими повідомленнями на схемі s38584 виконується значно повільніше інших. Цей алгоритм передбачає розсилання великої кількості нульових повідомлень. Наприклад, процес P1 має загальну кількість вихідних зв'язків, яке дорівнює 302, а вхідних – 311.

Моделювання на непарному числі процесорів може сповільнити роботу моделювання. Пов'язано це з підключенням нового обчислювального вузла до кластеру й додаткових витрат на пересилання по комп'ютерній мережі. На рис. 14 приведена залежність часу моделювання схеми s1196 від числа логічних процесів на послідовності у 10000 повідомлень.

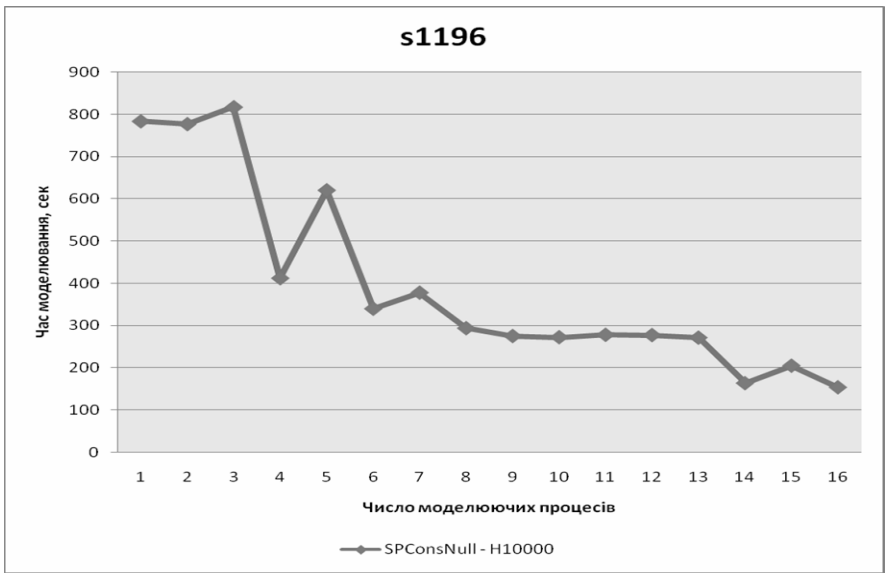


Рисунок 14 – Моделювання схеми s15850

Число подій, які створюються під час моделювання для розташування їх у локальному списку подій, для всіх протоколів моделювання має практично однакове значення. Зі збільшенням числа моделюючих процесів ця величина незначно зменшується.

Загальне число оновлень сигналів на виходах функціональних елементів майже однакове для будь-якого числа моделюючих процесів. Відмінність може зустрічатися тільки для оптимістичного протоколу, де можливі відкоти логічних процесів і повторна обробка сигналів.

Загальне число оброблених подій має більше значення для консервативного протоколу з нульовими повідомленнями (рис. 15). Обумовлено це передачею нульових повідомлень.

Найбільша кількість MPI-посилок необхідна для консервативного алгоритму з нульовими повідомленнями, що й обумовлює швидкість його роботи (рис. 16).

Максимальна кількість відкотів під час роботи оптимістичних протоколів була отримана на 10 процесах для схеми s15850. (рис. 17).

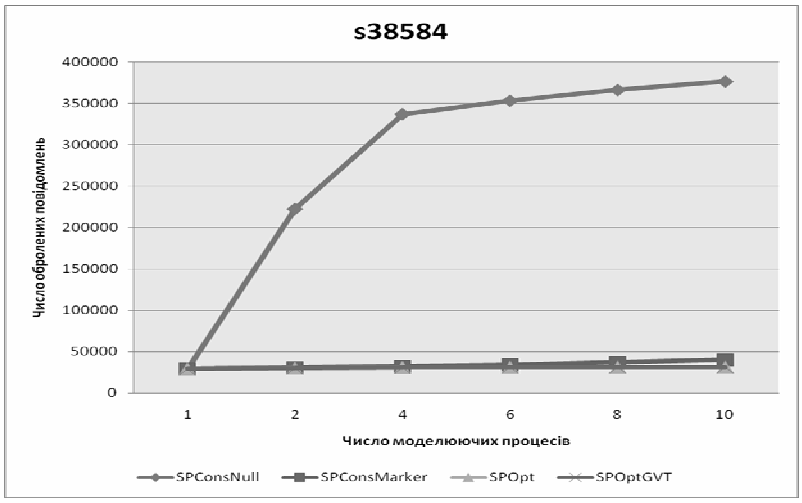


Рисунок 15 – Залежність числа оброблених повідомлень від числа процесів

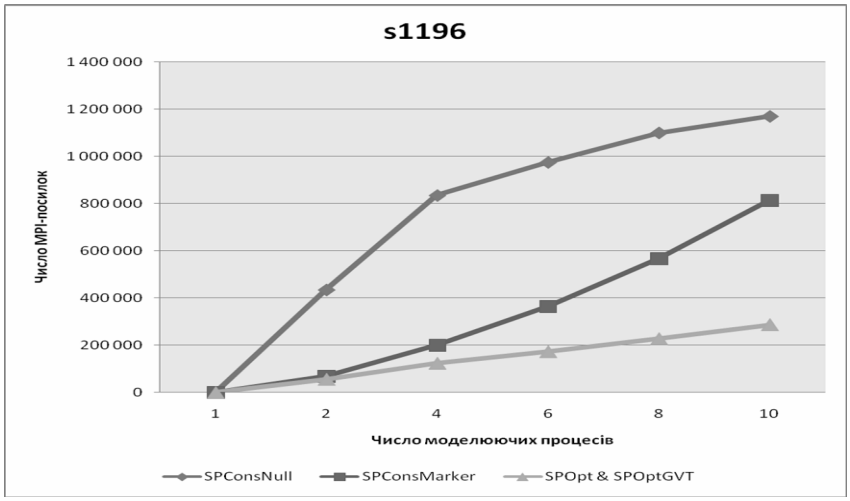


Рисунок 16 – Залежність числа MPI-посилок від числа процесів

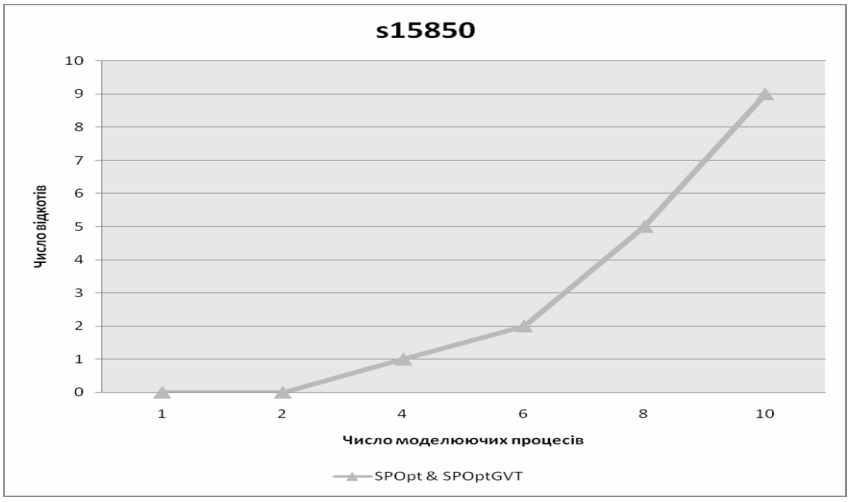


Рисунок 17 – Залежність числа відкотів від числа процесів

За результатами моделювання для схем з каталогу ISCAS-89 було отримане прискорення часу моделювання на 10 процесорах у порівнянні з послідовним моделюванням на 80% (рис. 18).

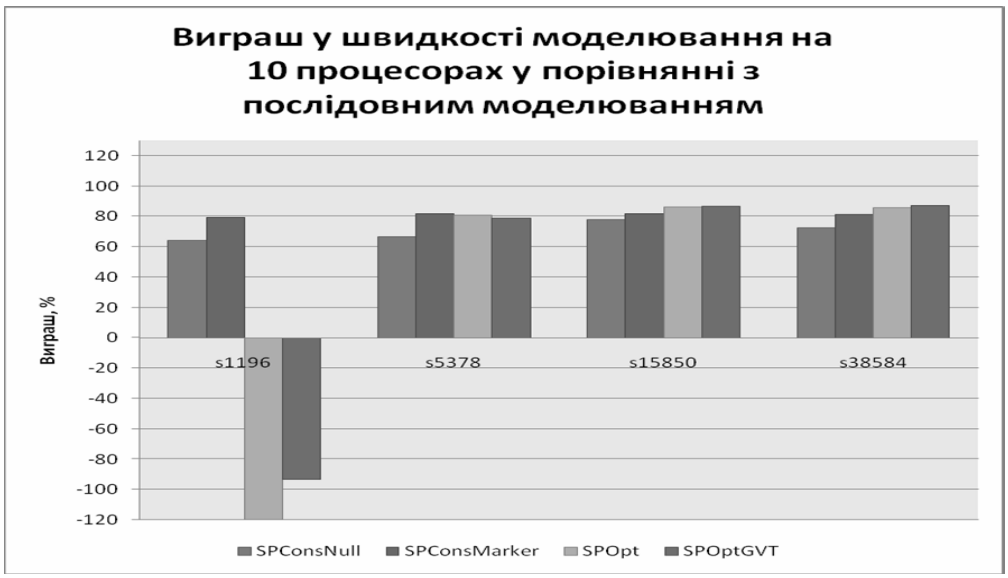


Рисунок 18 – Виграш у швидкості моделювання на ISCAS-схемах

Слід також зазначити, що час моделювання логічної схеми зі зростанням числа процесів зменшується не тільки за рахунок паралельних обчислень, але й за рахунок зменшення часу завантаження логічних схем і вхідних послідовностей, тому що кожний моделюючий процес завантажує тільки частину вхідних даних.

Висновки

Існує багато програмних систем, які дозволяють здійснювати моделювання. Спільним недоліком цих систем є те, що вони мають обмежену швидкодію, через послідовний характер виконання. Одним з можливих вирішень проблеми прискорення моделювання великих проектів є використання розподіленого імітаційного моделювання на обчислювальному кластері.

Аналіз існуючих систем розподіленого моделювання показав, що вони не забезпечують користувача всіма необхідними сервісами для проведення безпосередньо розподіленого моделювання, або працюють не на високопродуктивному обчислювальному кластері.

У ході виконання даної роботи були отримані нижченаведені результати.

1) Вперше розроблене програмне забезпечення для розподіленого моделювання цифрових логічних схем під керуванням обчислювального кластера Microsoft Windows Compute Cluster Server 2003. Передача

повідомлень між вузлами кластера реалізована за допомогою протоколу MPI. У програмній системі реалізовано 4 алгоритми синхронізації: консервативний з нульовими повідомленнями, консервативний з маркерами, оптимістичний без глобального віртуального часу й оптимістичний із глобальним віртуальним часом. Система дозволяє формувати звіти й збирати статистику про хід моделювання.

2) Проведені дослідження продуктивності розподіленої системи моделювання у залежності від протоколів синхронізації та розмірності схем моделювання. Досліджування виконувалися на схемах з міжнародного каталогу ISCAS-89.

3) Зроблений порівняльний аналіз протоколів синхронізації. Результати роботи розподіленої системи вказують на значне зменшення часу моделювання у порівняння з часом послідовного моделювання. Для цифрових схем із каталогу ISCAS-89 отримано прискорення моделювання на 10 процесорах у порівнянні з послідовним моделюванням на 80%.

4) Доведено, що значення часу моделювання залежить від схеми моделювання, якості розрізування, послідовності вхідних подій, загального часу моделювання й можливих затримок у комп'ютерній мережі.

Література

1. Окольнишников В.В. Представление времени в имитационном моделировании. Вычислительные технологии. Том 10, №5, Сибирское отделение РАН, 2005. – с. 57-77.
2. Charlie Russel. Overview of Microsoft Windows Compute Cluster Server 2003.
3. Астахов А.В., Ладыженский Ю.В. Методика развертывания Windows Compute Cluster Server 2003. Информатика и компьютерные технологии – 2007. // Материалы III научно-технической конференции молодых ученых и студентов. – Донецк, ДонНТУ – 2007. – с. 77-78.
4. Астахов О.В., Ладиженський Ю.В. Архітектура та планування завдань у Windows Compute Cluster Server 2003 для розподіленого моделювання. СНКПМІ-2008. // Матеріали одинадцятої всеукраїнської студентської наукової конференції з прикладної математики та інформатики. – Львів, ЛНУ ім. І.Франка, – 2008. – с.25-26.
5. Діденко Д. Г. Агент реплікації в розподіленій дискретно-подійній системі імітаційного моделювання OPENGPSS. Матеріали міжнародної наукової конференції "Інтелектуальні системи прийняття рішень та прикладні аспекти інформаційних технологій", 2006. – с. 264–266.
6. Allen R., Garlan D., Ivers J. Formal modeling and analysis of the HLA component integration standard. // Proc. of the 6th ACM SIGSOFT Intern. Symp. on Foundations of Software Engineering. 1998. – pp. 70–79.

7. Ferscha Alois. Parallel and Distributed Simulation of Discrete Event Systems. In Hardbound of Parallel and Distributed Computing. McGraw-Hill, 1995. – pp. 1003–1041.
8. Richard M. Fujimoto. Distributed Simulation Systems. In Proceedings of the 2003 Winter Simulation Conference S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds. – pp. 124-134.
9. В. Бигдан, Т Марьяновия, М Сахнюк. От последовательных к распределенным технологиям в имитационном моделировании. // В трудах первой всероссийской научно-практической конференции ИММОД-2003, ФГУП ЦНИИ технологии судостроения, Санкт-Петербург. 2003. – Том 1. – с. 59-63.
10. Jefferson D.R. Virtual time II: storage management in distributed simulation. // Proc. of the Ninth Annual ACM Symposium on Principles of Distributed Computing. 1990. – pp. 75–89.
11. Корнеев В.Д. Параллельное программирование в MPI. Новосибирск: Изд-во СО РАН, 2000.
12. В.А. Гришагин, А.Н. Свистунов. Параллельное программирование на основе MPI. Уч. пособие – Нижний Новгород: Изд-во ННГУ им. Н.И. Лобачевского, 2005. – с. 93.
13. Астахов А.В., Ладыженский Ю.В. Распределенное моделирование цифровых логических схем на вычислительном кластере. Материалы IV научно-технической конференции студентов, аспирантов и молодых ученых. – 25-27 ноября 2008, ДонНТУ, Донецк – 2008. – с. 83-85.
14. Ладыженский Ю.В., Попов Ю.В. Программная система для распределенного событийного логического моделирования дискретных цифровых устройств. Режим доступу: http://sim.1024.info/text/science/20051215.html
15. Ладыженский Ю.В., Попов Ю.В. Объектно-ориентированная модель протоколов синхронизации при распределенном логическом моделировании цифровых устройств // Наукові праці Донецького національного технічного університету. Випуск 64. – Донецьк: Вид-во ДонНТУ, 2003. – 280с. – с. 212–221.
16. Chandy K.M., Misra J. Asynchronous Distributed Simulation via a Sequence of Parallel Computations. Communications of the ACM, 24(11): 198–206, November, 1981.
17. Chandy K.M., Misra J. Distributed simulation: a case study in design and verification of distributed programs. // IEEE Transactions on Software Engineering. 1978. Vol. SE–5(5). – pp. 440–452.
18. Brglez F., Bryan D., Kozminski K. Combinational Profiles of Sequential Benchmark Circuits. ISCAS'89 Benchmark Circuits. – Proc. IEEE Int. Symposium on Circuits and Systems. – May 1989. – pp. 1929–1934.

Астахов Алексей Вячеславович.
В 2008 получил диплом магистра по специальности «Программное обеспечение автоматизированных систем» в Донецком национальном техническом университете.
Научные интересы: моделирование, параллельные вычисления.

Ладыженский Юрий Валентинович.
Кандидат технических. наук, доцент кафедры прикладной математики и информатики Донецкого национального технического университета.
Научные интересы: высокопроизводительные системы, параллельные и распределенные вычисления.

Дата надходження до редакції 26.12.2008 р.