ВИЯВЛЕННЯ ВЗАЄМНИХ БЛОКУВАНЬ В MPI ПРОГРАМАХ

Валерій Гавриленко[1], Олександр Галкін[1], Оксана Ковальчук[1], Анатолій Обшта[2]

[1]Національний технічний університет України "Київський політехнічний інститут", Київ, Україна
[2]Національний університет "Львівська політехніка", 79013, вул. Митрополита Андрея 5, Львів, Україна, 4-й корпус, к. 218

DEADLOCK DETECTION IN MPI PROGRAMS

Valery Gavrilenko[1], Oleksandr Galkin[1], Oksana Kovalchuk[1], Anatoliy Obshta[2]

[1]National Technical University of Ukraine "Kiev Polytechnic Institute", Kyiv, Ukraine
[2]National University "Lviv Polytechnic", 79013, Metropolit Andrey st., 5, Lviv, Ukraine

АНОТАЦІЯ. *Досліджено способи виявлення ситуацій, за яких використання блокуючих та деяких неблокуючих двоточкових процедур, а також колективних процедур може призвести до взаємного блокування. Наведено основні способи використання програми MPI-CHECK при виявленні блокувань. Показані приклади використання стратегії «рукостискання» для блокуючих та неблокуючих двоточкових процедур, а також колективних процедур MPI.*

Ключові слова: *блокування, комунікатор, ідентифікатор, процедура «рукостискання».*

АННОТАЦИЯ. *Исследованы способы выявления ситуаций, при которых использование блокирующих и некоторых неблокирующих двухточечных процедур, а также коллективных процедур может привести к взаимной блокировке. Приведены основные способы использования программы MPI-CHECK при обнаружении блокировок. Показаны примеры использования стратегии «рукопожатия» для блокирующих и неблокирующих двухточечных процедур, а также коллективных процедур MPI.*

Ключевые слова: *блокировка, коммуникатор, идентификатор, процедура «рукопожатия».*

SUMMARY. Purpose. *This paper focuses on the introduction of new technology MPI. The problem of identifying reciprocal interlocks in MPI programs.* ***Methodology/approach.*** *Approaches are investigated for identifying of situations in which the use of blocking and non-blocking two-point procedures and collective procedures can lead to deadlock. The main ways of using of the program of MPI-CHECK in deadlock detection are presented. The examples of using the strategy of "handshaking" for blocking and non-blocking two-point procedures and collective procedures of MPI are presented.* ***Findings.*** *The result of work is to identify mutual locks for non-blocking two-point and collective MPI procedures.* ***Research limitations/implications.*** *In this paper are considered the main ways to detect when using blocking and non-blocking some two-point procedures and collective procedures can happen deadlock. However, the program MPI-CHECK Version 1.0 does not detect situations which may occur deadlock, because these methods are used in MPI-CHECK Version 2.0 for MPI programs written in Fortran 90 format and Fortran 77..* ***Originality/value.*** *This data methods can be applied to MPI programs written in C or C ++.*

Key words: *deadlock, communicator, identifier, "handshaking" procedure.*

Вступ

Дана стаття присвячена впровадженню на кафедрі інформаційних систем і технологій Національного транспортного університету новітньої технології MPI в новому нормативному курсі "Технології розподілених систем та паралельних обчислень" для студентів, що навчаються за напрямом "Комп'ютерні науки". Зокрема, в якості об'єкта дослідження, розглядається проблема виявлення взаємних блокувань в MPI програмах.

Інтерфейс передачі повідомлень (The Message-Passing Interface, MPI) широко використовується під час написання програм для комп'ютерів із паралельно розподіленою пам'яттю. Оскільки написання програм, що використовують MPI, є досить складним і займає багато часу, то для полегшення цієї задачі була розроблена програма MPI-CHECK 1.0 [1]. Вона автоматично виконує такі завдання, як перевірка типу аргументу, перевірка обмежень для буферів повідомлень. Однак, MPI-CHECK версії 1.0 не виявляє ситуацій, в яких може трапитися взаємне блокування. Тому для виявлення випадків, коли при використанні блокуючих та деяких неблокуючих двоточкових процедур, а також колективних процедур може трапитися взаємне блокування, була розроблена програма MPI-CHECK версії 2.0.

MPI-CHECK 2.0 виявляє фактичні та потенційні взаємні блокування в MPI програмах. Фактичне блокування виникає в тому випадку, коли процес очікує завершення дії, яка ніколи не завершиться. Наприклад, фактичне взаємне блокування виникне тоді, коли процес виконає синхронне відправлення даних MPI *mpi_ssend* та не одержить відповіді від процедури отримання. Потенційне взаємне блокування може виникнути в тому ж випадку, що й фактичне, але в залежності від реалізації MPI. Наприклад, потенційне

блокування виникає тоді, коли процес виконує стандартне MPI відправлення *mpi_send* та не отримає відповіді, однак виклик *mpi_send* копіює повідомлення в буфер і виконання продовжується. Потрібно зауважити: якби процедура mpi_*send* не скопіювала повідомлення в буфер, то могло б виникнути фактичне взаємне блокування. Використовуючи децентралізований підхід, MPI-CHECK 2.0 виявляє фактичні та потенційні взаємні блокування при використанні блокуючих та деяких неблокуючих двоточкових процедур, а також колективних процедур [2].

Виклад основного матеріалу

MPI забезпечує як блокуючі, так і неблокуючі двоточкові процедури комунікації. Нагадаємо, що коли процес виконує блокуючу двоточкову процедуру, виконання не продовжиться, доки не буде безпечною процедура змінення буферу відправки/отримання.

Існує три категорії ситуацій, коли при використанні блокуючих двоточкових MPI процедур може виникнути взаємне блокування:

1. Процес виконує процедуру отримання, але немає відповідного виклику до процедури відправлення;

2. Процес виконує процедури *mpi_send*, *mpi_ssend* або *mpi_rsend*, але немає відповідного виклику до процедури отримання;

3. Внаслідок неправильного використання процедур відправлення та отримання може виникнути замкнутий цикл відправлень-отримань.

Очевидно, що ситуація, описана в першому випадку, спричинить фактичне взаємне блокування. Як було сказано вище, у ситуації, описаній в другому випадку, фактичне взаємне блокування виникне при використанні синхронного відправлення *mpi_ssend*, а також при використанні стандартного відправлення *mpi_send*. При використанні процедури *mpi_send*, ситуація, наведена в другому випадку, спричинить потенційне взаємне блокування. Відповідно до стандарту MPI, операція відправки по готовності *mpi_rsend* може розпочатися тільки тоді, коли отримання уже є ініціалізованим; в іншому випадку, операція являється помилковою і результат є невизначеним. Якщо процес викликає процедуру буферизованого відправлення *mpi_bsend* та не існує відповідного отримання, то це не являється ні фактичним, ні потенційним блокуванням. На даний момент MPI-CHECK не перевіряє відповідне отримання за викликом процедури *mpi_bsend*.

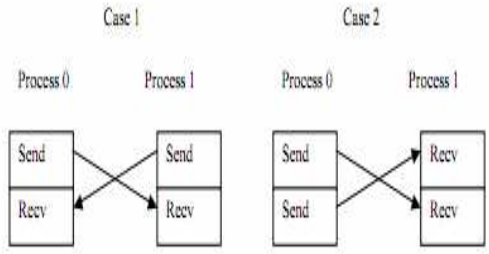

Рис.1.Цикл залежності з двома процесами
Fig.1. The cycle depends on two processes

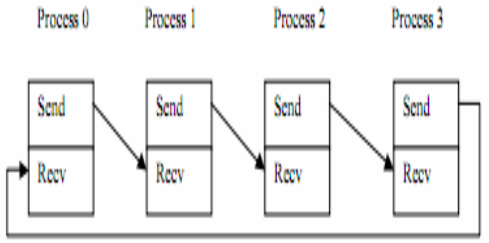

Рис.2.Цикл залежності з чотирма процесами
Fig.2. The cycle of dependency with four processes

На рис. 1 та 2 продемонстровано некоректне використання відправлень та отримань при використанні процедури *mpi_ssend* для циклу з двома та з чотирма процесами. Зауважимо, що відправлення не може завершитися, поки не буде надіслане відповідне отримання. Отже, така ситуація викличе фактичне взаємне блокування. Якщо в випадках, показаних на рис.1 та рис.2, використати процедуру *mpi_send*, то може відбутися або фактичне, або потенційне блокування, залежно від реалізації процедури та від розміру повідомлення. При використанні процедури *mpi_bsend*, взаємне блокування не відбувається, а, отже, таке використання є правильним [3].

Далі ми обговоримо способи, що можуть бути використані для автоматичного

виявлення фактичних та потенційних взаємних блокувань, описаних вище. Один з можливих способів – автоматична заміна всіх викликів *mpi_send* та *mpi_rsend* в MPI програми на *mpi_ssend* за допомогою MPI-CHECK. При запуску модифікованої програми під контролем відладчика, він буде автоматично зупинятися в точці взаємного блокування. Однак, цей підхід має декалька проблем. По-перше, на машині, що використовується в даний момент, може не існувати паралельного відладчика. По-друге, якщо такий відгадчик існує, то компіляція та запуск великого програмного коду можуть виконуватися неприпустимо довго. Тому цей спосіб не використовується в програмі MPI-CHECK.

Інший можливий спосіб виявлення фактичних та потенційних взаємних блокувань – запуск MPI програми під контролем центрального менеджера. Однак, при його використанні можуть виникнути ускладнення. Наприклад, припустимо, що при відладці програми використовується *p* процесів і на одному з них виконується центральний менеджер. Якщо блокування трапляється саме за тим процесом, на якому виконується центральний менеджер, тоді він припиняє функціонування. Також зауважимо, що центральний менеджер може значно затримувати MPI зв'язок в процесі роботи. Таким чином, для виконання MPI програми з *p* процесами, потрібно було б використовувати *p+1* процес.

Стратегія «рукостискання», що використовується в MPI-CHECK, може бути описана наступним чином. Частина стратегії включає в собі порівняння даних від виклику процедур відправлення та отримання. Якщо MPI-CHECK знаходить виклик процедури

mpi_send(buf, count, datatype, dest, tag, comm, ierror).

тоді в змінній send_info буде збережена наступна інформація:

send_info = {filename, start-line, end-line, count, get_rank(comm), datatype, tag},

Процедура «рукостискання» для *mpi_send* працює таким чином: процес, що виконує процедуру *mpi_send*, посилає *send_info* до процесу *dest*, використовуючи неблокуюче відправлення *mpi_isend* із унікальним ідентифікатором *MPI_CHECK_Tag1 + tag* (для уникнення конфлікту з іншими повідомленнями). Далі можуть виникнути наступні випадки:

1. Процес *dest* не отримав повідомлення, і процес, що надсилав повідомлення, не отримав відповіді протягом встановленого часу. В даному випадку видається попередження.

2. Процес *dest* отримав повідомлення, інформація в *send_info* є узгодженою з інформацією у виклику до *mpi_recv* і процес *dest* відправляє відповідь до процесу, що надсилав повідомлення про те, що все гаразд. Відповідь отримується за допомогою виклику *mpi_irecv*.

3. Процес *dest* отримав повідомлення, але інформація в *send_info* не є узгодженою з інформацією у виклику *mpi_recv*. В цьому випадку, процес *dest* видає повідомлення про присутність неузгодженості та відправляє відповідь до процесу, що надсилав повідомлення про його отримання.

Процедура «рукостискання» для *mpi_recv* працює наступним чином: процес, що виконує процедуру *mpi_recv*, чекає на отримання (викликаючи *mpi_irecv*) повідомлення від *source* з ідентифікатором MPI_CHECK_Tag1 + tag, де *tag* (ідентифікатор) отримується від виклику до *mpi_recv*. Якщо *tag* являється *mpi_any_tag*, тоді *mpi_any_tag* використовується як ідентифікатор для отримання повідомлення. Далі можуть виникнути наступні випадки:

1. Повідомлення не було отримано протягом встановленого часу. В даному випадку видається попередження.

2. Повідомлення було отримано та інформація в *send_info* є узгодженою з інформацією у виклику до *mpi_recv*. До процесу, що надсилав повідомлення, відправляється відповідь про те, що все гаразд.

3. Повідомлення було отримано, але інформація в *send_info* не є узгодженою з інформацією у виклику до *mpi_recv*. В цьому

випадку видається попередження та до процесу, що надсилав повідомлення, надсилається відповідь про його отримання.

Якщо виклик процедури *mpi_recv* використовує *mpi_any_source* і/або *mpi_any_tag*, може трапитися, що цей виклик може отримати відповідь від іншої процедури *mpi_send*, а не від тієї, з якою відбувалося «рукостискання». Для уникнення цієї проблеми MPI-CHECK змінює оригінальну процедуру mpi_recv із

call mpi_recv(buf, count, datatype, source, tag, comm, status, ierror)

на

call mpi_recv(buf, count, datatype, send_rank, send_tag, comm, status, ierror)

де *send_rank* та *send_tag* беруться з *send_info* під час «рукостискання». Також зазначимо, що в цій ситуації MPI-CHECK виявить взаємне блокування тільки в порядку виконання MPI програми [4].

Програма MPI-CHECK працює так, що користувачі мають можливість або зупинити виконання MPI програми при виявленні фактичного або потенційного взаємного блокування, або дозволити MPI-CHECK продовжити виконання MPI програми після виявлення проблеми. Ці параметри можна налаштувати, використовуючи змінну

ABORT_ON_DEADLOCK.

Виконання наступної команди

setenv ABORT_ON_DEADLOCK true

спричинить зупинку виконання програми (за допомогою виклику процедури *mpi_abort*) при виявленні фактичного або потенційного взаємного блокування.

Фактичне та потенційне взаємне блокування також може бути спричинене некоректним використанням процедури *mpi_probe*. Нагадаємо, що *mpi_probe* дозволяє визначити параметри вхідного повідомлення для того, щоб визначити, як його отримати. Оскільки *mpi_probe* є блокуючою процедурою, у випадку, коли відсутнє відповідне відправлення, може виникнути фактичне блокування процесу, що виконує *mpi_probe*. Ця ситуація ускладнюється тим, що одне відправлення може задовольнити декілька викликів до процедури *mpi_probe*.

Для виявлення фактичних або потенційних взаємних блокувань можна ще застосовувати процедуру *mpi_probe*, але з деякими змінами. Якщо відповідна процедура *mpi_probe* знаходиться перед викликом отримання, то процедура «рукостискання» повинна бути вставлена перед *mpi_probe*. Якщо таких процедур *mpi_probe* існує більше, ніж одна, то процедура «рукостискання» повинна бути вставлена перед першою з них. Таким чином, потрібно вирішити проблему, чи існує відповідний запит перед будь-яким викликом *mpi_probe* або *mpi_recv* (*mpi_sendrecv, mpi_sendrecv_replace*) в MPI програмі, та куди потрібно вставити процедуру «рукостискання».

Спочатку, аналогічно з використанням процедури «рукостискання» для *mpi_recv*, вставляємо її перед викликом *mpi_probe*. Однак зауважимо, що це може спричинити неправильну роботу процедури «рукостискання». Для уникнення цієї проблеми, MPI-CHECK зберігає список усіх викликів до *mpi_probe* з унікальним ідентифікатором {*communicator, tag, source*}. При вставці коду перед викликом *mpi_recv* та *mpi_probe* проводиться перевірка, чи {*communicator, tag, source*} співпадає із записами в цьому списку. Якщо так, то «рукостискання» пропускається; в іншому випадку воно виконується.

Існує неблокуюча версія процедури *mpi_probe*, що називається *mpi_iprobe*. Для її використання не потрібно ніяких «рукостискань», оскільки вона не викликає взаємних блокувань [5].

Виявлення взаємних блокувань для неблокуючих двоточкових MPI процедур

MPI також дозволяє використання неблокуючих двоточкових процедур, а саме *mpi_isend, mpi_issend, mpi_ibsend, mpi_irsend* та *mpi_irecv*. Неблокуючі відправлення/отримання можуть бути зіставлені з блокуючими отриманнями/ відправленнями.

Завершення неблокуючих відправлень та отримань вказується викликами процедур *mpi_wait*, *mpi_test*, *mpi_waitany*, *mpi_testany*, *mpi_waitall*, *mpi_testall*, *mpi_waitsome* та *mpi_testsome*. Наразі MPI-CHECK робить перевірки лише для викликів процедур *mpi_wait* та *mpi_waitall*. Якщо для позначення завершення будуть використані інші процедури, MPI-CHECK не перевірить завершення та в деяких випадках може навіть некоректно виявити помилки. Якщо існують неблокуючі виклики відправлення або отримання без відповідних викликів процедур *mpi_wait* або *mpi_waitall*, MPI-CHECK повідомляє користувачу про необхідність додати ці виклики.

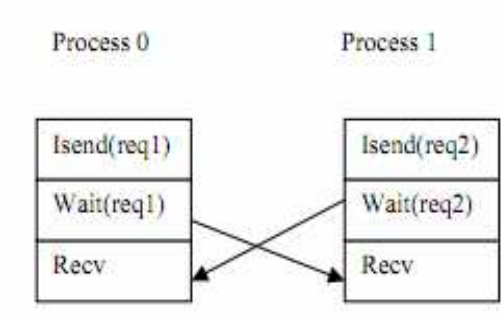

Рис.3. Цикл залежності, що включає неблокуючі виклики
Fig.3. Cycle dependence includes not blocking calls

Як і у випадку з блокуючими відправленнями та отриманнями, при використанні неблокуючих відправлень та отримань можуть виникнути фактичні та потенційні взаємні блокування [6]. Вони можуть трапитися під час виклику процедур *mpi_wait* або *mpi_waitall*, але не під час виклику неблокуючих відправлень чи отримань. Наприклад, взаємне блокування виникне, якщо не існує відповідного відправлення чи отримання. Також при використанні неблокуючих процедур можуть трапитися цикли залежності, як показано на рис.3.

На рис.4. показана ситуація, коли взаємне блокування ніколи не виникне внаслідок опису специфікації MPI. На рис.5. показана ситуація, за якої може виникнути потенційне взаємне блокування.

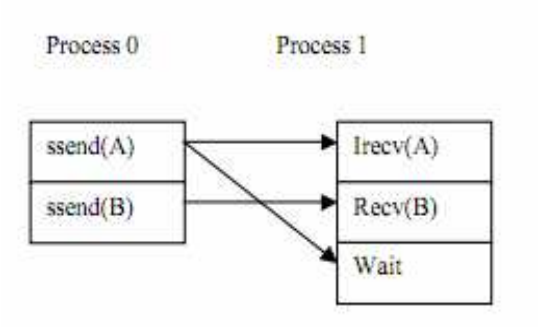

Рис.4. Ситуація, за якої ніколи не виникне взаємне блокування
Fig.4. The situation in which there will never deadlock

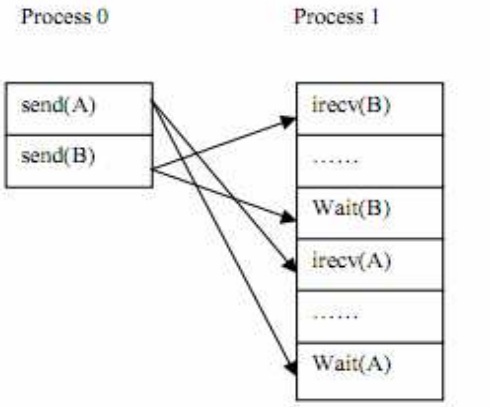

Рис.5. Ситуація, де може виникнути потенційне взаємне блокування
Fig.5. A situation may arise where a potential deadlock

MPI-CHECK виявляє фактичні та потенційні взаємні блокування при використанні неблокуючих процедур, використовуючи згадувану стратегію «рукостискання», подібну до тієї, що використовується для блокуючих процедур. При виявленні неблокуючого відправлення або отримання, MPI-CHECK вставляє код перед викликом, що ініціює «рукостискання», але не чекає його завершення. Код, що завершує «рукостискання», вставляється перед викликом відповідної процедури *mpi_wait* (або *mpi_waitall*).

За допомогою вставки коду до виклику процедури *mpi_finalize*, програма MPI-CHECK здатна визначити, чи існують незавершені виклики до неблокуючих процедур відправлення чи отримання. Ця ситуація виникає тільки в тому випадку, коли не було зроблено відповідного виклику до *mpi_wait*. MPI-CHECK відстежує всі неблокуючі виклики та їх

відповідні запити за допомогою запису інформації в глобально доступний масив *MPI_CHECK_Ireq* розміром *MAX*.

Виявлення взаємних блокувань для колективних MPI процедур

На відміну від двоточкової комунікації, колективні процедури бувають тільки блокуючими. Існують наступні випадки виникнення фактичних та потенційних взаємних блокувань при використанні колективних процедур:

1. Колективна процедура не викликається всіма процесами в комунікаторі;

2. Всі процеси в комунікаторі можуть не викликати відповідні колективні процедури в тому ж порядку;

3. Невірне впорядкування двоточкових та колективних процедур.

На рис.6 показана ситуація, що описана в першому випадку, коли два процеси виконують *mpi_gather*, в той час як третій – ні. Зауважимо, що в цій ситуації може виникнути фактичне або потенційне блокування, залежно від наявності синхронізації після виклику процедури *mpi_gather*.

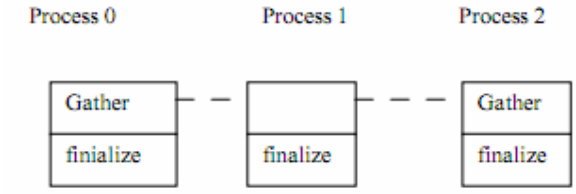

Рис.6. Колективна процедура з пропущеним викликом

Fig.6. Collective procedure with missed calls

На рис.7 показана ситуація, що описана в другому випадку. Стандарт MPI вимагає, що для правильної роботи програми потрібно викликати колективні зв'язки. Тоді, незалежно від того, є вони синхронізуючими чи ні, взаємне блокування ніколи не виникне.

Для колективних процедур, на відміну від двоточкових процедур, використовується *p* процесів. Цей метод полягає у використанні процесу 0 для запису інформації про всі інші процеси. Під час «рукостискання» всі ненульові процеси посилають ненульовому процесу в масив *send_info* наступну інформацію:

send_info = {file_name, start_line, end_line, get_rank(comm), call_name},

де *call_name* є ім'ям колективної процедури. Кожен такий виклик для уникнення конфліктів з іншими викликами має унікальний ідентифікатор *MPI_CHECK_Tag3*. Далі можуть виникнути наступні випадки:

1. Процес 0 не отримав повідомлення, і процес, що надсилав повідомлення, не отримав відповіді протягом встановленого часу. В даному випадку видається попередження.

2. Процес 0 отримав повідомлення, але змінна *call_name* в процедурі *send_info* відрізнялася від аналогічної змінної у виклику до процесу 0. В цьому випадку process 0 видає попередження та робота програми зупиняється незалежно від режиму її виконання.

3. Процес 0 отримав повідомлення та змінна *call_name* в процедурі *send_info* не відрізнялася від аналогічної змінної у виклику до процесу 0 В цьому випадку процс 0 надсилає відповідь з інформацією, що все гаразд, за допомогою виклику *mpi_irecv* із ідентифікатором *MPI_CHECK_Tag4*. Робота програми продовжується далі.

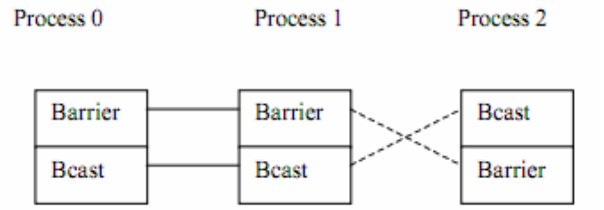

Рис.7. Неправильно впорядковані колективні процедури

Fig.7. Collective procedure the incorrectly ordered

Для нульового процесу процедура «рукостискання» працює наступним чином: процес 0 виконує *p-1* викликів до процедури *mpi_irecv* з ідентифікатором *MPICHECK_Tag3*, намагаючись отримати *p-1* копію *send_info*, що була надіслана ненульовими процесами. Можуть трапитися наступні випадки:

1. Було отримано *p-1* копій *send_info* та в кожній з них змінна *call_name* співпадала зі змінною *call_name* нульового процесу. В цьому випадку до кожного з *p-1* процесів

надсилається відповідь з інформацією, що все гаразд.

2. Хоча б одна з копій *send_info* не була отримана протягом встановленого часу. В даному випадку видається попередження.

3. Одна з *p-1* копій of *send _info* була отримана, але змінна *call_name* в надісланій інформації не співпадає зі змінною *call_name* в нульовому процесі. В цьому випадку видається попередження про невідповідний виклик та виконання програми зупиняється [7-10].

Зауважимо, що способи виявлення взаємних блокувань для колективних процедур, що наведені вище, можуть бути застосовані лише для зв'язків в межах єдиної групи процесів (інтра-зв'язок), але не для розділеної групи зв'язків (інтер-зв'язок).

На відміну від MPI 1.0, що дозволяє використовувати для колективних процедур тільки інтракомунікатори, MPI 2.0 підтримує також і використання інтеркомунікаторів. Для виявлення фактичних та потенційних взаємних блокувань при використанні інтеркомунікаторів MPI-CHECK виконує наступні дії. Спочатку для визначення типу комунікатора використовується MPI процедура *mpi_comm_test_inter(comm, flag).* Далі потрібно врахувати наступні зміни:

1. Нульовий процес в локальній та віддаленій групах збирає інформацію від всіх ненульових процесів у віддаленій групі.

2. Для відповідних нульового та ненульового процесів, у віддаленій групі проводиться операція «рукостискання».

Висновки

Отже, підводячи підсумки, підкреслимо, що в даній статті були розглянуті основні способи виявлення випадків, коли при використанні блокуючих та деяких неблокуючих двоточкових процедур, а також колективних процедур може трапитися взаємне блокування. Однак, програма MPI-CHECK версії 1.0 не виявляє ситуацій, в яких може трапитися взаємне блокування, тому ці методи застосовуються в MPI-CHECK версії 2.0 для MPI програм, написаних в форматі Fortran 90 та Fortran 77. Також дані способи можуть бути застосовані до MPI програм, написаних мовою C або C++.

References

1. *Message Passing Interface* Forum. MPI: A Message-Passing Interface Standard, 1995.
2. *Message Passing Interface* Forum. MPI-2: Extensions to the Message-Passing Interface, 1997.
3. *Luecke G., Chen H., Coyle J., Hoekstra J., Kraeva M., Zou Y.,* 2002. MPI-CHECK: A tool for checking Fortran 90 MPI programs. Concurrency and Computation. Practice and Experience.
4. *Vetter J.S., Supinski B.R.,* 2000. Dynamic software testing of MPI applications with Umpire. Proceedings of SC2000. Dallas, November.
5. *MPI* - The Complete Reference, Volume 2. The MPI Extensions, Gropp et al, MIT Press, 1999.
6. *MPI* - The Complete Reference, Volume 1. The MPI Core, 2nd edition, Snir et al, MIT Press, 1999.
7. *Gottlieb A., George S.,* 1989. Highly parallel computing. Redwood City, Calif., Benjamin Cummings.
8. *Roosta, Seyed H.,* 2000. Parallel processing and parallel algorithms : theory and computation. New York, NY, Springer.

Надійшла до редакції
04.12.2012 р.

Затверджена до друку
10.12.2012 р.