УДК 004.254(045)
Мельник В.М.к.ф-м.н. доц., Багнюк Н.В. к.т.н. доц., Мельник К.В. к.т.н. доц.
Луцький національний технічний університет

АЛГОРИТМИ ПЛАНУВАННЯ ВІДОМИМ ОБМІНОМ ДАНИХ В ОПЕРАЦІЙНИХ СИСТЕМАХ ДЛЯ МУЛЬТИСОКЕТНИХ БАГАТОЯДЕРНИХ СЕРВЕРІВ

Мельник В.М., Багнюк Н.В., Мельник К.В. Алгоритми планування відомим обміном даних в операційних системах для мультисокетних багатоядерних серверів. Для основних виробників чіпів відомі всі новітні багатоядерні мікропроцесори. Мультисокетні системи, які побудовані на базі таких процесорів зазвичай використовуються для виконання різних серверних додатків. Типічно кожен процесор в такій системі розділяє кеш-пам'ять або на рівні L2 або на рівні L3. В залежності від програми, яка виконується в системі, передача кеш-пам□ять-кеш-пам'ять (КП-КП) між сокетами може впливати на загальну продуктивність системи. Дана робота представляє нову оптимізацію планування операційної системи (ОС) з метою зменшення впливу подібної міжсокетної передачі між кеш-пам□яттю і кеш-пам'яттю.

Спостерігаючи передачу між кеш-пам□яттю і кеш-пам'яттю поміж кожною парою ниток для кожного кванта планування та застосування чотирьох різних алгоритмів було придумано новий графік потоків для наступного запланованого кванта. Цей новий графік потенційно скорочує міжсокету передачу між кеш-пам□яттю і кеш-пам'яттю для наступного кванта планування. Досліджувався вплив цих алгоритмів на 18 реальних тестах. За допомогою тестів вивчалася міжсокетна передача між кеш-пам'яттю і кеш-пам'яттю, де спостерігалось зменшення більш як на 99,3% на деяких тестах і, в середньому, від 5,5% до 24% в залежності від використовуваного алгоритму планування.

Ключові слова: алгоритми планування, кеш-пам□ять, передача КП-КП, продуктивність, управління процесами планування, багатопроцесорний сервер, нитки.

Мельник В.М., Багнюк Н.В., Мельник Е.В. Алгоритмы планирования известным обменом данных в операционных системах для мультисокетных многоядерных серверов. Для основных производителей чипов известны все новейшие многоядерные процессоры. Мультисокетни системы, построенные на базе таких процессоров обычно используются для выполнения различных серверных приложений. Типично каждый процессор в такой системе разделяет кэш-память или на уровне L2 или на уровне L3. В зависимости от программы, которая выполняется в системе, передача кэш-память-кэш-память (КП-КП) между сокетами может влиять на общую производительность системы. Данная работа представляет новую оптимизацию планирования операционной системы (ОС) с целью уменьшения воздействия подобной мижсокетнои передачи между кэш-памятью и кэш-памятью.

Наблюдая передачу между кэш-памятью и кэш-памятью между каждой парой нитей для каждого кванта планирования и применения четырех различных алгоритмов было придумано новый график потоков для следующего запланированного кванта. Этот новый график потенциально сокращает передачу между кэш-памятью и кэш-памятью между сокетами для последующего кванта планирования. Исследовалось влияние этих алгоритмов на 18 реальных тестах. С помощью тестов изучалась миждусокетная передача между кэш-памятью и кэш-памятью, где наблюдалось уменьшение более чем на 99,3% в некоторых тестах и, в среднем, от 5,5% до 24% в зависимости от используемого алгоритма планирования.

Ключевые слова: алгоритмы планирования, кэш-память, передача КП-КП, производительность, управление процессами планирования, многопроцессорный сервер, нити.

Melnyk V.M., Bagnyk N.V., Melnyk K.V. Data knowlage exchange algorithms using in the multi-soket operating systems for multicore servers. Major chip manufacturers have all introduced multi-core microprocessors. Multi-socket systems built from these processors are routinely us for running various server applications. Typically, each rocessor in such a system shares a cache at either the L2 or L3 level. Depending on the application that runs on the system, intersocket cache-to-cache transfers can affect overall performance. This paper presents a new operating system (OS) scheduling optimization to reduce the impact of such inter-socket cache-to-cache transfers.

By observing the pattern of cache-to-cache transfers between every pair of threads for each scheduling quantum and applying four different algorithms, we come up with a new schedule of threads for the next quantum. This new schedule potentially cuts down the inter-socket cache-to-cache transfers for the next scheduling quantum. We studied the impact of these algorithms on 18 real-world benchmarks. For the benchmarks we studied, inter-socket cache-to-cache transfers are down by as much as 99.3% on some benchmarks and, on average, between -5.5% and 24% depending on the scheduling algorithm employed.

Keywords: Scheduling Algorithms, cache memory, cache memory, cache-to-cache transfers, performance, process scheduling management, multiprocessor server, threads.

Постановка наукової проблеми. Багато комерційних серверних додатків сьогодні працюють на мультисокетних багатоядерних серверах. Ці програми, як правило, страждають від різних видів недостатків кешу. Більшість з цих недоліків можна уникнути, збільшивши або розмір кешу, кеш асоціативність або розмір рядка. Однак є й інші види недоліків зокрема так звані недоліки в комунікації, практично недоліки міжсокетної комунікації, які притаманні додаткам і не можуть бути легко вирішені шляхом обмежування параметрів кешу. Це залежить від кількості

спільного використання програми та шляху, яким додаток був написаний. Щоб покращити цю проблему, один із способів – це переписати програму. Інший спосіб полягає в спостереженні за предметами спільного використання та адаптації планувальника завдань операційної системи (ОС), щоб мінімізувати вплив недоліків зв□язку між сокетами. У цій роботі представлено таку оптимізацію планування ОС, яка відстежує сокетні трансфери КП-КП для кожного кванта планування, а потім застосовує чотири різні алгоритми вирішення, де слід запланувати кожен потік для наступного кванта планування. Встановлено, що для різних тестів, які вивчалися, різні алгоритми і їх зміни зменшували міжсокетну КП-КП передачу, в середньому, від 5,5 до 24% і до 99,3% для деяких тестів. Наскільки відомо, жодна з поточних комерційних ОС не оптимізовує власний планувальник завдань з урахуванням недоліків зв□язку, то далі розмова піде про алгоритми планування, їх складність і ймовірність, з якою вони вносять вклад в додатки, а також методика використання для оцінки цих алгоритмів та результати роботи.

Аналіз досліджень. Алгоритми планування. Різні алгоритми планування можуть допомогти зменшити міжсокетну КП-КП передачу. Розглянемо один простий і три складні комплексні алгоритми в цій роботі. Для всіх алгоритмів ми виділяємо той, що відповідає лічильнику підтримки апаратної продуктивності, для розрахунку міжсокетної КП-КП передачі. Для кожного ядра ми призначаємо 480-бітний регістр, розділений на п□ятнадцять 32-бітних пакетів між ниткою, що виконується на даному ядрі і ниток, які протікають на всіх інших п□ятнадцяти ядрах (наша максимальна мікропроцесорна (МП) конфігурація – це система з чотирма сокетами, де кожен з сокетів об□єднює чотири ядра). Ми можемо зменшити ширину регістра шляхом спрощення КП-КП передач, замість того, щоб рахувати кожну передачу [1]. У кожному запланованому кванті для кожного запиту зчитування, що впливає на КП-КП передачу, ведеться спостереження, яка з ниток пересилає дані, і збільшується лічильник, що відповідає цій нитці. Наприкінці запланованого кванта ОС зчитує лічильники для кожної нитки, а потім застосовує один з чотирьох алгоритмів для прийняття рішення про планування для наступного кванта планування. Ця інформація лічильника може бути збережена в ОС як частина контексту нитки. Алгоритми описуються в наступних підрозділах.

Послідовність вирішення проблеми

Проблема, яку потрібно вирішити, можна сформулювати наступним чином. Розглянемо 16 ниток, що виконуються на системі з чотирма сокетами, де кожен з сокетів об□єднює чотири ядра. Представимо орієнтований граф з 16 вершинами, де кожна з вершин представляє ядро. Ребро з вершини *i* в вершину *j* представляє КП-КП передачу як результуючу пердавання даних від процесора *i* до процесора *j* (див. рис. 1). Крайове навантаження *Wij* являє собою кількість КП-КП передач. Проблема полягає в тому, щоб згрупувати разом ці вершини з максимальною сумою країв так, щоб глобальний максимум суми ребер був максимальний. Ми представляємо простий алгоритм, а потім більш складний покращений алгоритм, з подальшим удосконаленням покращеного алгоритму, а потім з невеликим штрихом поліпшення, так що він може скоро вийти. Зверніть увагу, що найбільш складний алгоритм намагається згрупувати нитки тільки на основі даного кванта планування. Це не може виявитися оптимальним для наступного етапу. Це надихає на думку про вивчення алгоритмів застосування для удосконалення найбільш складного алгоритму.

Виклад основного матеріалу й обґрунтування отриманих результатів дослідження.

Алгоритм 1. У цьому алгоритмі вибирається одна нитка для кожного сокета, що дасть найбільший вклад шляхом переходу до наступного сокета. Наприклад, якщо взяти систему з чотирма сокетами, то вирішуємо заздалегідь, що переміщаємо першу нитку від сокета 0 до сокета 1, іншу нитку – з сокета 1 в сокет 2, наступну нитку – з сокета 2 в сокет 3, а ще наступну – з сокета 3 до сокета 0. Отже рух між сокетами вирішено заздалегідь. Для кожного сокета розраховуємо, яка з чотирьох ниток дасть найбільший вклад шляхом переходу до наступного сокета. Після цього вибираємо і переміщаємо цю конкретну нитку в наступний квант планування. В кінці кожного кванта планування (крім першого) обчислюється вклад, отриманий від нової "карти " розподілу ниток в порівнянні з вихідною і представляється цей вклад в результаті запропонованого алгоритму.

Складність

$O(p*N*K) + O(p*N) = O(p*N*K)$

p = номер кванта планування; *N* = Кількість ниток; *K* = кількість ядер на сокет (4 в нашому випадку)

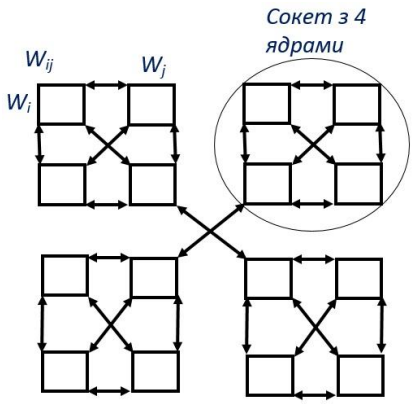

Рис. 1. Кластерний граф (міжкластерний рух не повністю показаний на графі)

Попередній номер складності, в основному, для того щоб розрахувати внесок для кожної з N ниток шляхом об□єднання його КП-КП передачі даних зі всіма K нитками свого власного сокета, і тими, що на наступному сокеті. Потім береться різниця між ними, щоб обчислити внесок, який можна отримати, переміщаючи цю нитку до наступного сокета. Це має бути зроблено для кожних фази/кванта планування. Все це стає *O(p*N*K)* (примітка: вся складність представляється для всіх квантів планування; альтернативно, можна видалити член умови *p* і спостерігати складність на кожен квант). Після розрахунку внеску потрібно порівняти внески від всіх *K* ниток у сокеті для того, щоб вирішити, яка нитка має максимальний внесок і спрямувати цю нитку до наступного сокета. Це має бути зроблено для кожного сокета і для кожного кванта планування. Це стає *O(p*N)*. Таким чином, в цілому, складність є *O(p*N*K)*. Цей алгоритм може бути легко розпаралелений. Якщо кожне ядро виконує частину алгоритму, то складність для кожного кванта планування/фази стає *O(K^2)*, де K є постійним (по суті, *O(1)*).

Варіант 1 алгоритму 1

Є два варіанти для першого алгоритму. Обидва варіанти пов□язані зі спостереженням фазової поведінки КП-КП передачі і дають відповідні переваги в його реалізації. У першому варіанті кластеризується певна кількість *L* квантів планування в один квант, а потім приймається рішення планування для наступної *L* кількості квантів планування. Для кожної нитки обчислюється вклад руху цієї нитки до наступного кластера для кожного з *L* квантів планування, а потім об□єднуються вклади для всіх *L* квантів. Виходячи з цього вкладу вибирається одна з чотирьох ниток в кластері, яка приносить найбільший вклад при переході на наступний кластер. Цей процес повторюється для кожного кластера (двох кластерів для двосокетної системи і чотирьох кластерів для системи з чотирма сокетами). Після цього отримується нове "відображення" ниток і використовується для наступних L квантів планування.

Складність

*O(p*H*K)+O((P/L)*N) = O(p*H*K)*;

p = кількість квантів планування; *N* = кількість ниток; *K* = кількість ядер на сокет (в нашому випадку 4); *L* = кількість планування квантів за якими ведеться спостереження одночасно для того, щоб прийняти рішення на наступні *L* квантів. Аргумент складногсті такий же, як вихідний, за винятком того, що число фаз спадає з коефіцієнтом *L* (на другий відрізок часу). Таким чином, загальна складність становить *O(p*H*K)*.

Варіант 2 алгоритму 1

У другому варіанті також ведеться спостереження за поведінкою в межах фази, але використовується "алгоритм вивчення", з метою дізнатися про які-небудь загальні закономірності, що відбуваються в *L* квантів планування, як описано вище. Отримується відображення ниток для кожного *L* кванта планування та формується для них бітова картина. Є стільки бітових карт, скільки є сокетів в системі. Якщо нитка *i* запланована на сокеті *0*, то біт встановлений у положенні *i* на бітовій карті відповіддає сокету *0*. Визначається, чи є загальний шаблон (загальний шаблон

зображений з двох логічно помножених AND послідовних бітових карт і число бітів, встановлених в результуючому шаблоні повинно бути більше ніж 2), що проходить від одного кванта до іншого в бітових картах. Якщо загальний шаблон виконується, то вклад буде *i*(1/L)* для цього шаблону, в якому *i* – це число кванта планування в L-розгалуженні квантів планування. Це дає більший ввклад для "останніх" шаблонів, що утворюються, і менший вклад для "старих" шаблонів. Також надається вклад *(i*1/(L*L))* для кожної з вихідних (операція заперечення AND) бітових карт, які отримуються для кожного з *L* квантів планування. Потім шаблони впорядковуються в залежності від вкладу, що присвоюється даному шаблону. Після сортування переглядається відсортований список і можна спробувати сформувати карту, використовуючи найбільш ті шаблони, що частіше зустрічаються. Отримана карта використовується для наступного L планування квантів.

Складність

*O(p*N*K)+O((p*N)+O((p/L)*((S^2*L+S*L)^2))+O((p/L)*(S^2*L+S*L))=O(p*N*K)+O((p/L)*((S^2*L+S*L)^2))*;

p = кількість квантів планування; *N* = кількість потоків; *K* = кількість ядер на сокет (4 в нашому випадку); *L* = кількість квантів планування, що спостерігається одночасно для прийняття рішення на наступні *L* квантів; *S* = кількість сокетів.

Перші дві умови (терми) є такими ж, як і в варіанті 1. Третя і четверта умови застосовуються для алгоритму вивчання. Для алгоритму вивчання виконується карта з урахуванням ідентифікаторів ниток, запланованих на кожний сокет (в бітовій карті встановлюється біт в положенні ID для нитки). Отримуються S-бітні карти, утворені для кожного кванта планування, де *S* – це кількість сокетів. Кожна з *S*-бітних карт логічно перемножується з *S*-бітними картами наступного кванта планування, щоб отримати відповідний шаблон, що приводить до *O(S^2)*. Крім цього, також включаються *S*-бітові карти для кожного кванта планування із вкладом *(I*1/(L*L))*, якбуло описано раніше, і це, в свою чергу, приводить до *O(S)*. Це повторюється для всіх L квантів планування за якими ведеться спостереження. Всі відповідні шаблони вставляються в відсортованому порядку і систематизується тип сортування для вставки. Таким чином, складність стає *O((S^2*L+S*L)^2)*. (Примітка: для простоти призначається тип вставки. Насправді, можна було б це робити за допомогою двійкового дерева пошуку, щоб відстежувати треки входу або які-небудь інші ефективні структури, і в цьому випадку алгоритм стає набагато ефективнішим.). Це повинно повторюватися для всіх *p/L* фаз і, таким чином, загальна складність стає *O((p/L)*((S^2*L+S*L)^2))*. Після такого введення в мульти-карту слід зробити лінійний хід від шаблонів узгодження до отримання нового відображення ниток. Це має бути зроблено для всіх *(p/L)* фаз, а результат очікується в четвертому діапазоні.

Алгоритм 2

Цей алгоритм є дещо складнішим в порівнянні з першим. В кінці кожного кванта збираються лічильники від усіх ядер, а потім вибираються всі можливі комбінації груп з чотирьох ниток *(N©4=N)* і підсумовується КП-КП передача між парами (там буде 12 можливих пар (напрямлених ребер) КП-КП передачі між чотирма нитками). Потім відсортовуються суми в порядку спадання. Після сортування, вибирається перша група з чотирьох ниток з найвищою сумою і спланується на сокет *0*, а потім вибирається наступна група з чотирьох ниток з високою сумою і спланується на сокет *1*, і так далі, поки не закінчиться планування ниток на всіх сокетах. Тим не менш, це не є оптимальною евристикою, тому що використовувався спрощений алгоритм для вибору групи з чотирьох ниток з відсортованого списку.

Складність

*O(p*N©4) + O(p*((N©4)^2)) + O(p*(N©4)) = O(p*((N©4)^2))*;

p = кількість квантів планування; *N* = кількість ниток; *N©4*=N-комбінованість 4.

Ці три умови є наступними. Перша умова відповідає за формування *N©4* парних взаємодій для всіх *p* квантів планування. Друга умова вставлює попарно суми в відсортованому порядку. Встановлюється сортування для вставки, яке повинно бути зроблено для кожного кванта планування. Отже, *O(p*((N©4)^2))*. Третя умова здійснює лінійну апроксимацію, щоб підібрати нове відображення на наступний квант планування в спрощеному варіанті. Це знову-таки має бути зроблено для всіх *p* квантів планування. Отже, *O(p*(H©4)*.

Варіант 1 алгоритму 2

Існують два варіанти другого алгоритму. Обидва вони пов□язані зі спостереженнями фазової поведінки КП-КП передачі і стимулюють його відповідні переваги.

У першому варіанті (аналогічно як для алгоритму 1), кластеризується певна кількість *L* квантів планування в один квант, а потім приймається рішення планування для наступної *L* кількості квантів планування. Для всієї можливої комбінації з чотирьох ниток *(N©4)* об□єднується попарно взаємодія для всіх *L* квантів планування, а потім нормалізується ця сума наступним чином:

*Нормована сума = (Сума / (max_pair_wise_interaction - min_pair_wise_interaction)) * (max_pair_wise_interaction);*

Після цього відсортується кожен можливий кластер відповідно до нормованої суми. Після сортування відсортований список переглядається, а потім формуються кращі два або чотири кластери (для дво- або чотирьох-сокетних систем) використовуючи спрощений описаний вище алгоритм.

Складність

*O((p/L)*N©4) + O((p/L)*((N©4)^2)) + O((p/L)*(N©4)) = O((p/L)*((N©4)^2))*;

p = кількість планування квантів; *N* = кількість потоків; *N©4* = N-комбінованість 4; *L* = кількість квантів планування за якими одночасно ведеться спостерження, щоб прийняти рішення на наступні *L* квантів. Ці три умови є такими ж, як в алгоритмі 2 без зміни. Зараз просто слід запустити алгоритм для *P/L* квантів планування, тому що ведеться спостереження за *L* квантами одночасно.

Варіант 2 алгоритму 2

У другому варіанті (за аналогією з алгоритмом 1) також ведеться спостереження за базовою фазовою поведінкою, але використовується "алгоритм вивчення" щоб дізнатися про загальні закономірності, що відбуваються в *L* квантах планування, як описано вище. Отримується карта ниток для кожного кванта планування за допомогою спрощеного алгоритму, а потім застосовується алгоритм вивчення (описаний в алгоритмі 1), щоб дізнатися про які-небудь загальні закономірності між відображеннями. Грунтуючись на цих загальних шаблонах отримується нове відображення, яке буде використовуватися для наступних *L* квантів планування.

Складність

*O(p*N©4)+O(p*((N©4)^2))+O(p*(N©4))+O((p/L)*((S^2*L+S*L)^2))+O((p/L)*(S^2*L+S*L))=O(p*((N©4)^2))+O((p/L)*((S^2*L+S*L)^2))*

p = кількість квантів планування; *N* = кількість ниток; *L* = кількість квантів планування, за якими ведеться спостереження одночасно для прийняття рішення на наступні *L* квантів; *S* = кількість сокетів.

Перші три умови є такими ж, як описано в основному алгоритмі і його варіанті. Четверта і п□ята умови пов□язані з алгоритмом вивчення і були пояснені в варіанті 2 алгоритму 1.

Алгоритм 3

Цей алгоритм є покращенням у порівнянні з алгоритмом 2. Тут в кінці кожного кванта відмічаються лічильники від усіх ядер, а потім вибираються всі можливі комбінації групи з чотирьох ниток *(N©4)* і підсумовується вся КП-КП передача між парами (буде сформовано 12 можливих пар (напрямлених ребер) КП-КП передачі між чотирма нитками). Потім відсортуються суми в порядку спадання. Після сортування переглядається відсортований список, утворюючи всі комбінації груп *M* кластерів (де *M* являє собою число сокетів в системі), а потім вибирається ця група для *M* кластера з максимальною сумою спареної взаємодії поміж усіма *M* кластерами. Це може не виявити кращу комбінацію спарення для сполучення для наступного кванта (так як на даному етапі відсутні знання про майбутнє, а тільки можна спостерігати минуле або базову фазову поведінку, щоб покращити умови).

Складність

O(p*N©4) + O(p*((N©4)^2)) + O(p*((N©4)^2)) =O(p*((N©4)^2));

p = кількість квантів планування; *N* = кількість ниток; *N©4* = N-комбінованість 4.

Перший доданок – це формування *N©4* парних взаємодій для всіх *p* квантів планування. Другий доданок вставляє попарні суми в відсортованому режимі. Самостійно встановлюється сортування вставки. Це має бути зроблено для кожного кванта планування. Отже, *O(p*((N©4)^2))*. Третя складова для перегляду відсортованого списку *(N©4-1)* рази, щоб отримати глобально

кращу групу *M* кластерів. Це має бути зроблено для кожного з *p* квантів планування. Отже, отримуємо доданок *O(p*((N©4)^2))*.

Варіант 1 алгоритму 3

Є два варіанти в цьому алгоритмі, як з алгоритмами 1 і 2. Обидва вони грунтуються на дослідженні базової фазової поведінки. У першому варіанті (за аналогією з алгоритмом 1) кластеризується певна *L* кількість квантів планування в один квант, а потім прийняття рішення планування для наступного *L* номера квантів планування. Для всіх можливих комбінацій з чотирьох ниток (*N©4*) слідує об□єднання парно у взаємодію для всіх *L* квантів планування, а потім ця сума нормалізується:

Нормована сума=(Сума/(max_попарн_вз - min_попарн_вз))(max_попарн_вз);*

Потім відсортовується кожен можливий кластер відповідно до нормованої суми. Після сортування переглядається відсортований список (*N©4-1* разів), а потім утворюються кращі два або чотири кластери (для дво- або чотирьох-сокетних систем) з використанням алгоритму 3.

Складність

*O((p/L)*N©4) + O((p/L)*((N©4)^2)) +O((p/L)*((N©4)^2)) = O((p/L)*((N©4)^2))*;

p = кількість квантів планування; *N* = кількість ниток; *L* = кількість квантів планування, *N©4* = N-комбінованість 4, *L* = кількість квантів планування, що спостерігається одночасно для прийняття рішення на наступні L квантів.

Ці три умови є так само, як і в алгоритмі 3, без зміни, але тепер потрібно стартувати алгоритм для (*P/L*) квантів планування, тому що спостереження ведеться одночасно за L квантами.

Варіант 2 алгоритму 3

У другому варіанті (за аналогією з алгоритмом 1) також досліджується базова поведінка фази, але використовується "алгоритм вивчення", щоб дізнатися, які загальні закономірності відбуваються в *L* кванті планування, як описано вище. Зображаємо нитки для кожного кванта планування з використанням алгоритму 3 (без зміни), а потім застосуємо алгоритм вивчення (описаний вище в алгоритмі 1), щоб дізнатися про які-небудь загальні закономірності між відображеннями. Опираючись на ці загальні моделі отримується нове відображення, яке буде використовуватися для наступних *L* квантів планування.

Складність

O(p*N©4)+O(p*((N©4)^2))+O(p*((N©4)^2))+O((p/L)*((S^2*L+S*L)^2))+O((p/L)*(S^2*L+S*L))=O(p*((N©4)^2))+O((p/L)*((S^2*L+S*L)^2))

p = кількість квантів планування, *N* = кількість ниток, *L* = кількість квантів планування, що спостерігається одночасно для прийняття рішення на наступні *L* квантів, *S* = кількість сокетів.Перші три умови є такими ж, як описано в основному алгоритмі і його варіанті представлення. Четвертий і п'ятий доданки пов□язані з алгоритмом вивчення, як описано в варіанті 2 алгоритму 1.

Алгоритм 4

Цей алгоритм є таким же, як алгоритм 3, але використовує невелику гнучкість для того, щоб вийти з алгоритму раніше, поки буде досягнутий такий же результат, як в алгоритмі 3. Коли здійснюється аналіз відсортованого списку, то відстежується доданок для максимальної попарної взаємодії для всіх комбінацій M кластерів проаналізованих досі (де M – кількість сокетів в системі). Коли досягається точка, в якій не прослідковується ніяка можливість перевищення максимуму, що спостерігався досі, то перед цим треба вийти з алгоритму. То ж, як швидко буде здійснено вихід з алгоритму – стає дуже залежним показником. Скільки буде зекономлено за рахунок швидкого виходу – подано в таблиці 4.

Складність

O(p*N©4) + O(p*((N©4)^2)) + O(p*((N©4)*E)) = O(p*((N©4)^2))

p = кількість квантів планування, *N* = кількість ниток, *N©4* = N-комбінованість 4, *E* = фактор передуючого виходу.

Перші два члени такі самі, як і в алгоритмі 3. Третій доданок для переходу до *E* раз відсортованого списку, для отримання глобальної кращої групи кластерів *M*. Коефіцієнт *E* дуже малий, часто менший за *0.2*N©4* (таблиця 4). Це має бути зроблено для кожного з *p* квантів планування. Отже, доданок набуває вигляду *O(p*((N©4)*E))*.

Варіант 1 алгоритму 4

Є два варіанти цього алгоритму. Обидва вони засновані на спостереженні базової фазової поведінки.

У першому варіанті (за аналогією з алгоритмом 1), ми кластеризуємо певне *L* число квантів планування в один квант, а потім приймаємо рішення планування для наступного *L* числа квантів планування. Для всіх можливих комбінацій з чотирьох ниток (*N©4*) ми об□єднуємо попарну взаємодію для всіх *L* квантів планування, а потім нормалізувати цю суму:

*Нормована сума = (сума / (max_ попарн_вз - min_ попарн_вз)) * (max_ попарн_вз);*

Потім відсортовується кожен можливий кластер відповідно до нормованої суми. Після сортування, аналізується відсортований список (*E* раз), а потім утворюється кращі два або чотири кластери (для дво- або чотирьох-сокетних систем) з використанням алгоритму 4.

Складність

*O((p/L)*N©4) + O((p/L)*((N©4)^2)) +O((p/L)*((N©4)*E)) = O((p/L)*((N©4)^2))*

p = кількість квантів планування, *N* = кількість ниток, *N©4* = N-комбінованість 4, *E* = фактор передуючого виходу, *L* = кількість квантів планування що спостерігається в один і той же час для прийняти рішення на наступні *L* квантів.

Ці три умови є так само, як в алгоритмі 4 без зміни, але слід стартувати алгоритм для (P/L) квантів планування, тому що спостереження ведеться за L-квантами одночасно.

Варіант 2 алгоритму 4

У другому варіанті (за аналогією з алгоритмом 1) також ведеться спостереження за базовою фазовоюповедінкою, але використовується "алгоритм вивчення" щоб дізнатися про будь-які загальні закономірності, що відбуваються в *L* квантів планування. Зображаються нитки для кожного кванта планування з використанням алгоритму 4 (без зміни), а потім застосується алгоритм вивчення (вже описаний), щоб дізнатися про які-небудь загальні закономірності між відображеннями. Опираючись на ці загальні підходи отримується нова карта на наступні L квантів планування.

Складність

*O(p*N©4)+O(p*((N©4)^2))+O(p*((N©4)*E))+O((p/L)*((S^2*L+S*L)^2))+O((p/L)*(S^2*L+S*L))=O(p*((N©4)^2)+O((p/L)*((S^2*L +S*L)^2))*

p = кількість квантів планування, *N* = кількість ниток, *N©4* = N-комбінованість 4, *E* = фактор передуючого виходу, *L* = кількість квантів планування що спостерігається в один і той же час для прийняти рішення на наступні *L* квантів, S = кількість сокетів.

Перші три умови є такими ж, як описано в основному алгоритмі і його варіанті. Четвертий і п□ятий доданки відображають алгоритм вивчення, що роз□яснюється в варіанті 2 алгоритму 1.

Методологія

Таблиця 1. Параметри тестування

Тестування	**К-сть сокетів**	**Довжина інструкції (в мільйонах)**	**Тестування**	**К-сть сокетів**	**Довжина інструкції (в мільйонах)**
Бізнес-обробка	2	74	SPEC WEB 05	4	140
Бізнес-обробка	4	149	Робочий стіл мультикористувачів	2	67
SPEC JBB 2000	2	145	Робочий стіл мультикористувачів	4	135
SPEC JBB 2000	4	145	робоче навантаження 1 для бази даних (БД)	2	72
SPEC JBB 2005	2	107	робоче навантаження 1 для БД	4	145
SPEC JBB 2005	4	120	робоче навантаження 2 для БД	4	145
SPEC JBB 2005-2	2	124	робоче навантаження 3 для БД	4	140
SPEC JBB 2005-2	4	124	робоче навантаження 4 для БД	2	68
SPEC WEB 05	2	70	робоче навантаження 4 для БД	4	136

Було використано внутрішній симулятор AMD для запуску 18 серверних слідів для дво- і чотирьох-сокетної конфігурації для збору статистики передачі КП-КП серед усіх пар ниток. Ці статистичні дані потім подаються в програму, яка імітує алгоритми планування, описані вище, і

дає загальний вклад (зниження відсотків у КП-КП передачі) в сумі досягнутих кожним алгоритмом. Критерії перераховано в таблиці 1, а параметри конфігурації – в таблиці 2. Допускається 5 мс кванти планування на процесорі частотою 2 ГГц. У середньому, моделювання проводилося для 28 квантів.

Таблиця 2. Параметри архітектури

Параметри архітектури	**Значення**
Кількість сокетів	2 або 4
Кількість ядер на сокет	4
Л1 ДКеш (на ядро)	64 КВ, 2-шляхи, 64-байтовий блок
Л2 Кеш (на ядро)	512 КВ, 16-шляхів, 64-байтовий блок
Л3 Кеш (спільний кеш – 1 на сокет)	6МВ, 48-шляхів, 64- байтовий блок
Протокол кеш-когерентності	MOESI
Мережа внутрішнього зв□язку	Лінія для 2 сокетів (сокет0 <-> сокет1), Квадрат для 4 сокетів (сокет0 <-> сокет1 <-> сокет2 <-> сокет3 <-> сокет0)

Результати

Результати для двохсокетної і чотирьохсокетної конфігурацій для всіх чотирьох алгоритмів наведені в таблиці 3. У позначенні, наприклад, A1pl, A – позначає алгоритм, 1 – позначає число алгоритмів, p – позначає базову фазову поведінку, і l – позначає алгоритм вивчення. Для базової фозової зміни спостерігаються одночасно два кванти планування для того, щоб прийняти рішення для наступних двох квантів.

Таблиця 3. Процентне зниження для міжсокетної КП-КП передачі на алгоритм

№ серії	**Тест**	**A1**	**A1p**	**A1pl**	**A2**	**A2p**	**A2pl**	**A3**	**A3p**	**A3pl**	**A4**	**A4p**	**A4pl**
1	Бізнес-обробка (2)	14	22	16	18	2	22	18	2	22	18	2	22
2	Бізнес-обробка (4)	35	37	34	47	19	49	47	19	49	47	19	49
3	SPEC JBB 2000 (2)	-0.7	-0.5	-0.5	1	-1	1	3	-1	3	3	-1	3
4	SPEC JBB 2000 (4)	1.6	0.7	-0.6	2	0	2	3	0	3	3	0	3
5	SPEC JBB 2005 (2)	5.9	2.5	2.5	12	6	12	15	6	15	15	6	15
6	SPEC JBB 2005 (4)	5.5	3.9	3.8	10	1	10	11	1	10	11	1	10
7	SPEC JBB 2005-2 (2)	6.9	0.3	0.6	6	-3.5	5.3	8.3	-3.3	8.8	8.3	-3.3	8.8
8	SPEC JBB 2005-2 (4)	9.3	8.4	9	10	-1	10.4	11	-1	11.1	11	-1	11.1
9	SPEC WEB 05 (2)	-13	-8.4	-7.1	-6	-24	-5.4	-2	-24	0	-2	-24	0
10	SPEC WEB 05 (4)	32	29	27	45	13.7	41.4	48	13.8	46	48	13.8	46
11	Робочий стіл мультикористувачів (2)	18	19	17.4	13.8	10.1	12.3	25	10.1	25	25	10.1	25
12	Робочий стіл мультикористувачів (4)	3.4	3	-14	15	-22	12.3	25	-19	25	25	-19	25
13	Робоче навантаж-я 1 для БД (2)	5.7	-11	-11	99.3	-12	99	99.3	-11	99	99.3	-11	99

14	Робоче навантаж-я 1 для БД (4)	42	38	28	99.2	-20	99	99.2	-18	99	99.2	-18	99.2
15	Робоче навантаж-я 2 для БД (4)	2	1.4	0.8	2.2	0.96	1.6	3.64	0.96	2.47	3.64	0.96	2.47
16	Робоче навантаж-я 3 для БД (4)	-21	-21	-21	0	-44	0	0	-38	0	0	-38	0
17	Робоче навантаж-я 2 для БД (4)	3.3	3.4	3.4	3	2.4	3.3	5.6	2.4	6.4	5.6	2.4	6.4
18	Робоче навантаж-я 4 для БД (4)	-11	-15	-16	3	-27	3.5	4.1	-26	4.98	4.1	-26	4.98
Середнє		7.7	6.2	4	21.1	-5.5	21	23.5	-4.8	23.88	23.5	-4.8	23.78

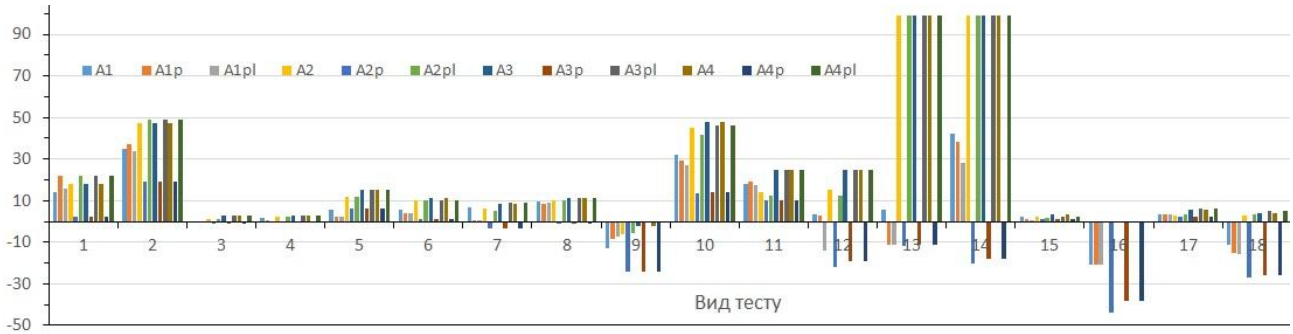

Рис. 2. Відсоткове зниження в міжсокетній КП-КП передачі на алгоритм (у тесті вісь X визначається за серійним номером у таблиці 3, а середнє значення не наводиться)

Для результатів, зображених на малюнку 2 (із таблиці 3) видно, що середній розмір вкладу від алгоритмів (відсоткове зниження в міжсокетній КП-КП передачі) коливається від 5,5% до 24% (будь ласка зверніть увагу, що в графіку не показано алгоритм 4, тому що вклад від нього по суті такий же, як від алгоритму 3). Отримано збільшення КП-КП передачі тому що спарювання знайдено за заданим алгоритмом для заданого числа фаз не може підтримувати наступні кілька фаз. Алгоритм 4 з використанням базового фазового алгоритму вивчення дає нам великий вклад. Алгоритм 2 (скорочений алгоритм) без базового фазового вивчення пропонує майже такий же великий вклад (21%), не будучи дуже складним. Є додатки, в яких для алгоритму вивчення передбачається (наприклад, бізнес-обробка, робоче навантаження 4 для БД з двосокетною та чотирьохсокетною конфігурацією), в середньому, алгоритм 2 (звичайна версія) який виконується досить добре. Робоче навантаження 1 для БД виділяється з максимальним вкладом 99,2%. За браком місця загальний вклад не перераховується під час виконання кожної програми. Але, як кращий варіант, для робочого навантаження 1 для БД (як двох- так і чотирьохсокетного) отримано приблизно вклад в розмірі 10% під час виконання.

Загалом, для малих серверів середньої дальності алгоритм 2 дає хороший компроміс між продуктивністю і складністю. Для великих серверів простий алгоритм може бути кращим вибором; він також надає додатковий вклад, що може бути реалізовано в розподіленій/паралельній формі.

У таблиці 4 ми представляємо відсоткове зниження числа ітерацій циклу, отриманих за рахунок виконання дострокового виходу в алгоритмі 4. Для більшості додатків зменшення відсотків спостерігається на рівнях 80-и або 90-а. Це говорить, що якщо ми хочемо вибрати кращий алгоритм виконання, то слід враховувати допомогу здійснення раннього виходу для реальних додатків.

Споріднені дослідження

Алгоритми кластеризації ниток були розглянуті Thekkath і Еггерс [2]. Їх дослідження базується на тому, щоб знайти кращий спосіб угруповання ниток, які з□єднюють області пам□яті на тому ж процесорі так, щоб максимізувати кеш-обмін і повторно його використати. Однак, шлях, якого вони дотримуються, є на базі адрес і може бути схильний до помилок, особливо в міграціях обміну. Якщо процеси P0, P1, P2 беруть участь в міграційному обміні, і якщо обирається

P0 і P1 щоб запланувати на той же сокет, а рядок кеш-пам□яті насправді мігрує від P0 до P2, а потім назад на P1, то зниження в міжсокетній КП-КП передачі не спостерігатиметься. У цій роботі відстежується обмін з використанням попарних лічильників і представляється найкращий глобальний алгоритм. В роботі припускається, що число ниток відповідає кількості процесорів, іншими словами, система завжди працює з балансуванням навантаження. Вивчення запропонованих алгоритмів також проходить на більш реальних тестуваннях на базі комерційних серверів.

Наша робота найбільше нагадує роботу, виконану авторами роботи [1], яка також пропонує використання лічильників продуктивності апаратного забезпечення для кластеризації ниток в багатопроцесорних системах з поєднаним використанням пам'яті. Вони використовували легкий алгоритм кластеризації і повідомили про зниження між віддаленими стійками кеша до 70%. У цій роботі використовуються чотири різних алгоритми планування і повідомляється про скорочення у віддалених недостатках кеш-пам'яті до 99,3%. Крім того, використовується інший підхід, щоб зібрати міжсокетні КП-КП передачі за допомогою лічильників продуктивності. Використання віддалених стійок кеша на діапазон адрес не вважається найкращим, так як це може приводити до проблеми спільного використання пам□яті, описаної вище, а замість цього підраховуються попарні передачі віддаленого кешу. Такий підхід працює для серверів малого і середнього радіусу дії.

Для великих машин (таких як Cray Red Storm, Aprro International Atlas [3], і т.д.), можна використовувати грубо- та дрібнокластерний підхід такий як поєднання двох досить близьких сокетів в один "логічний сокет". Це пов□язано з тим, що, навіть якщо не можна запланувати високовзаємодіючі нитки на одному і тому ж сокеті, то можна запланувати їх на близьких сокетах, щоб зменшити кількість переходів. Припускається, що базовий вузол для всіх КП-КП передач для обраного кластера ниток попадає в цей сокет. Один із способів вирішення цієї проблеми є підтримання в кожному лічильнику головного вузла підрахунку КП-КП передач між кожною парою ниток, а потім взяття основного вузла до уваги для реалізації алгоритму планування.

Таблиця 4. Процентне зниження в ітераціях за алгоритмом 4

Тестування	**К-сть сокетів**	**% спадання (в циклічних підрахунках)**	**Тестування**	**К-сть сокетів**	**% спадання (в циклічних підрахунках)**
Бізнес-обробка	2	96	Робочий стіл мультикористувачів	2	86
Бізнес-обробка	4	99	Робочий стіл мультикористувачів	4	99
SPEC JBB 2000	2	50	Робоче навантаження 1 для бази даних (БД)	2	97
SPEC JBB 2000	4	73	Робоче навантаження 1 для БД	4	99.8
SPEC JBB 2005	2	61	Робоче навантаження 2 для БД	4	88
SPEC JBB 2005	4	77	Робоче навантаження 3 для БД	4	99
SPEC JBB 2005-2	2	75	Робоче навантаження 4 для БД	2	71
SPEC JBB 2005-2	4	91	Робоче навантаження 4 для БД	4	97.4
SPEC WEB 05	2	84			
SPEC WEB 05	4	98			

В роботі [4] автори досліджували підходи, щоб виявити блокування обміну в просторі користувача між мультинитковими додатками за допомогою бібліотек анотування синхронізації на рівні користувача. На базі цієї інформації нитки, використовуючи те ж блокування, мігрують на

тому ж процесорі. Дана робота проводиться в тому ж дусі, але на більш загальному рівні, який може застосовуватися до будь-якого виду обміну області пам'яті.

Багато вчених досліджували підтримку ОС для мінімізації конфлікту кеш-пам□яті і проблеми пропускної здатності розподілених кеш-процесорів L2/L3 [5,6,7]. Однак дана робота спрямована на зменшення впливу недостатків зв'язку, які притаманні додаткам і може бути доповненням до цих робіт.

Висновки та перспективи подальшого дослідження. Багато комерційних серверних додатків сьогодні працюють на мультисокетних багатоядерних серверах. Такі програми, як правило, терплять проблеми від міжсокетних недостатків зв□язку, що зменшує їх загальну продуктивність. В роботі представлено операційну систему оптимізації з метою зниження наслідків недостатків міжсокетного зв□язку. Спостерігаючи міжсокетну КП-КП передачу між різними нитками і додаткові включення для рішень планування ОС, міжсокетні недостатки зв□язку для деяких додатків значно знижуються. З чотирьох різних алгоритмів вивчених на 18 реальних тестах, простий алгоритм зменшує міжсокетні недостатки зв□язку аж до 42% (в середньому = 7,7%) і варіації з трьох інших складних алгоритмів зменшують міжсокетну взаємодію до 99,3 % (в середньому – від 21% до 24%). В цілому, для малих і середніх серверів, спрощений алгоритм пропонує хороший компроміс між продуктивністю і складністю. Для великих серверів спрощений алгоритм може бути кращим вибором між продуктивністю та складністю, а простий алгоритм може бути кращим вибором.

Дана робота може бути поліпшена в декількох напрямках. Можна використовувати базову фазову поведінку на основі дослідження останньої L фази для прийняття рішення планування тільки для наступної фази замість наступних L фаз. Можна також використовувати кращі алгоритми машинного вивчення щоб дізнатися про будь-які зміни фазової поведінки між квантів планування і включити це в рішення планування. Загалом, для будь-якого довгострокового додатка з стійкими закономірностями, апаратне забезпечення може забезпечити зворотний зв□язок з ОС, яка в свою чергу може використовувати цю інформацію для адаптації своєї політики на користь продуктивності додатків.

1. Tam D., Azimi R., Stumm M. Thread Clustering: Sharing-Aware Scheduling on SMP-CMPSMT Multiprocessors. In ACM SIGOPS Operating System Review. June 2007.
2. Thekkath R., Eggers S.J. Impact of sharing based thread placement on multi-threaded architectures. In Int-l. Symp. on Computer Architecture. 1994.
3. www.top500.org
4. Sridharan S. *et al.* Thread migration to improve synchronization performance. In Workshop on Operating System Interference in High Performance Applications, 2006.
5. Nakajima J. *et al*. Enhancements for Hyper-Threading technology in the operating system – seeking the optimal micro-architectural scheduling. In Intl. Parallel and Distributed Processing Symp. 2005.
6. Snavely A. *et al*. Symbiotic job scheduling for a simultaneous multithreading processor. In Conf. On Architectural Support for Programming Languages and Operating Systems, 2000.
7. El-Moursy *et al.* Compatible phase coscheduling on a CMP of multi-threaded processors. In Intl. Parallel and Distributed Processing Symp. 2006.