7. Технологія OpenMP

7.1. Спільна пам'ять. Хто розробляє стандарт?

7.2. Основи OpenMP.

7.3. Прагми OpenMP.

OpenMP дозволяє легко і швидко створювати багатопоточні додатки на алгоритмічних мовах Fortran і C/C++. При цьому директиви OpenMP аналогічні директивам препроцесора для мови C/C++ і є аналогом коментарів у алгоритмічній мові Fortran. Це дозволяє в будь-який момент розробки паралельної реалізації програмного продукту при необхідності повернутися до послідовного варіанту програми.

7.1 Спільна пам'ять

OpenMP - це стандартна модель для паралельного програмування в середовищі зі спільною пам'яттю. У даній моделі всі процеси спільно використовують загальний адресний простір, до якого вони асинхронно звертаються із запитами на читання і запис. У таких моделях для управління доступом до загальної пам'яті використовуються різні механізми синхронізації типу семафорів та блокування процесів. Перевага цієї моделі з точки зору програмування полягає в тому, що поняття монопольного володіння даними відсутнє, отже, не потрібно явно задавати обмін даними між потоками, що їх задають, та потоками, що їх використовують. Ця модель, з одного боку, спрощує розробку програми, але, з іншого боку, ускладнює розуміння і управління локальністю даних, написання детермінованих програм. В основному вона використовується при програмуванні для архітектур зі спільною пам'яттю.

Прикладами систем зі спільною пам'яттю, які мають велике число процесорів, можуть служити суперкомп'ютери Cray Origin2000 (до 128 процесорів), HP 9000 V-class (до 32 процесорів в одному вузлі, а в конфігурації з 4 вузлів - до 128 процесорів), Sun Starfire (до 64 процесорів).

Хто розробляє стандарт?

Розробкою стандарту займається організація OpenMP ARB (ARchitecture Board), до якої увійшли представники найбільших компаній - розробників SMP-архітектур і програмного забезпечення. Перша версія специфікації OpenMP (www.openmp.org) з'явилася в 1997 році і призначалася для мови програмування Фортран. Біля витоків OpenMP стояли такі відомі компанії, як IBM, Intel, Sun і Hewlett-Packard. У 1998 році з'явилися варіанти OpenMP для мов C/C++, і на даний момент останньою є версія 3.0 [5].

В даний час OpenMP підтримується більшістю розробників паралельних обчислювальних систем: компаніями Intel, Hewlett-Packard, Silicon Graphics, Sun,

IBM, Fujitsu, Hitachi, Siemens, Bull і іншими. Багато відомих компаній в галузі розробки системного програмного забезпечення також приділяють значну увагу розробці системного програмного забезпечення з OpenMP. Серед цих компаній відзначимо Intel, KAI, PGI, PSR, APR, Absoft і деякі інші. Значне число компаній і науково-дослідних організацій, які розробляють прикладне програмне забезпечення, в даний час використовує OpenMP при розробці своїх програмних продуктів. Серед цих компаній і організацій відзначимо ANSYS, Fluent, Oxford Molecular, NAG, DOE ASCI, Dash, Livermore Software.

Компілятори gcc і gfortran мають вбудовану підтримку паралелізації OpenMP починаючи з версії 4.2. Для використання цієї можливості слід додати ключ `-fopenmp` при компіляції. OpenMP версії 3.0 підтримується починаючи з версії gcc 4.4. OpenMP 2.5 підтримується також компілятором Microsoft Visual C + + 2005 (слід використовувати ключ /openmp) і компіляторами Intel C або Intel Fortran починаючи з версії 10.1 (ключ -openmp).

7.2 Основи OpenMP

Будь-яка програма, послідовна або паралельна, складається з набору областей двох типів: послідовних областей і областей розпаралелювання. При виконанні послідовних областей породжується тільки один головний потік виконання (процес). У цьому потоці ініціюється виконання програми, а також відбувається її завершення. У послідовної програми цей потік є єдиним протягом виконання всієї програми. У паралельній програмі в областях розпаралелювання породжується цілий ряд паралельних потоків. Породжені паралельні потоки можуть виконуватися як на різних процесорах, так і на одному процесорі обчислювальної системи. В останньому випадку паралельні процеси (потоки) конкурують між собою за доступ до процесора. Управління конкуренцією здійснюється планувальником операційної системи за допомогою спеціальних алгоритмів. В операційній системі Linux планувальник завдань здійснює обробку процесів за допомогою стандартного карусельного (round-robin) алгоритму. При цьому тільки адміністратори системи мають можливість змінити або замінити цей алгоритм системними засобами. Таким чином, у паралельних програмах в областях розпаралелювання виконується ряд паралельних потоків. Принципова схема паралельної програми зображена на рис. 9.1.

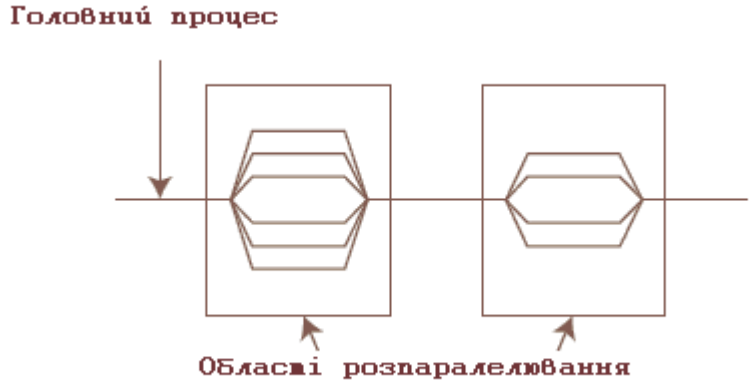

Рис. 9.1. Принципова схема паралельної програми

При виконанні паралельної програми робота починається з ініціалізації і виконання головного потоку (процесу), який у міру необхідності створює і виконує паралельні потоки виконання, передаючи їм необхідні дані. Паралельні потоки з однієї паралельної області програми можуть виконуватися як незалежно один від одного, так і з пересиланням та отриманням повідомлень від інших паралельних потоків. Остання обставина ускладнює розробку програми, оскільки в цьому випадку програмістові доводиться займатися плануванням, організацією і синхронізацією посилки повідомлень між паралельними потоками. Таким чином, при розробці паралельної програми бажано виділяти такі області розпаралелювання, в яких можна організувати виконання незалежних паралельних потоків. Для обміну даними між паралельними процесами (потоками) в OpenMP використовуються загальні змінні. При зверненні до загальних змінних у різних паралельних потоках можливе виникнення конфліктних ситуацій при доступі до даних. Для запобігання конфліктів можна скористатися процедурою синхронізації. При цьому треба мати на увазі, що процедура синхронізації - дуже дорога операція по тимчасових витратах і бажано по можливості уникати її або застосовувати якомога рідше. Для цього необхідно дуже ретельно продумувати структуру даних програми.

Виконання паралельних потоків в паралельній області програми починається з їх ініціалізації. Вона полягає у створенні дескрипторів породжуваних потоків і копіюванні всіх даних з області даних головного потоку в області даних паралельних потоків, що створюються. Ця операція надзвичайно трудомістка - вона еквівалентна приблизно трудомісткості не менше 1000 машинних команд. Ця оцінка надзвичайно важлива при розробці паралельних програм с допомогою OpenMP, оскільки її ігнорування веде до створення неефективних паралельних програм, які виявляються часто повільнішими ніж їх послідовні аналоги. Справді: для того щоб отримати виграш у швидкодії паралельної програми, необхідно, щоб трудомісткість паралельних процесів в областях розпаралелювання програми істотно перевершувала б трудомісткість породження паралельних потоків. В іншому випадку ніякого виграшу за швидкодією отримати не вдасться, а часто можна опинитися навіть і в програші.

Після завершення виконання паралельних потоків управління програмою знову передається головному потоку. При цьому виникає проблема коректної передачі даних від паралельних потоків головного. Тут важливу роль грає синхронізація завершення роботи паралельних потоків, оскільки в силу цілого ряду обставин час виконання навіть однакових за трудомісткістю паралельних потоків непередбачувано (воно визначається як історією конкуренції паралельних процесів, так і поточним станом обчислювальної системи). При виконанні операції синхронізації паралельні потоки, вже завершили своє виконання, простоюють і чекають завершення роботи самого останнього потоку. Природно, при цьому неминуча втрата ефективності роботи паралельної програми. Крім того, операція синхронізації має трудомісткість, порівнянну з трудомісткістю ініціалізації паралельних потоків.

Які переваги OpenMP дає розробнику?

1. За рахунок ідеї **"інкрементального розпаралелювання"** OpenMP ідеально підходить для розробників, хто хоче швидко розпаралити свої обчислювальні програми з великими циклами. Розробник не створює нову паралельну програму, а просто послідовно додає в текст послідовної програми OpenMP-директиви.
2. При цьому, OpenMP - досить **гнучкий механізм**, що надає розробнику великі можливості контролю над поведінкою паралельного програми.
3. Передбачається, що OpenMP-програма на однопроцесорній платформі може бути використана **як послідовна** програма, тобто немає необхідності підтримувати послідовну і паралельну версії. Директиви OpenMP просто ігноруються послідовним компілятором, а для виклику процедур OpenMP можуть бути підставлені заглушки (stubs), текст яких наведений у специфікаціях [17].

Далі наведено короткий опис специфікації OpenMP та наведено приклади використання в програмах на алгоритмічних мовах C і Фортран. Матеріал включає короткий опис основних директив OpenMP, процедур і змінних середовища. Для вивчення всіх тонкощів використання OpenMP слід звертатися до керівництва користувача [17] та книг [1-4].

7.3 Прагми OpenMP

Специфікація OpenMP визначає набір прагм. Прагма - це директива компілятора що вказує, як обробляти код, який слідує за нею. Найбільш суттєвою є прагма `#pragma omp parallel`, що визначає область паралельності.

Коли прагми OpenMP використовуються в програмі, вони дають вказівку компілятору, що підтримує OpenMP, створити виконуваний модуль, який буде виконуватися паралельно з використанням декількох потоків. При цьому в вихідний код необхідно внести достатньо невеликі зміни. Прагми OpenMP дозволяють використовувати красивий, одноманітний і переносимий інтерфейс для паралелізації програм на різних архітектурах та системах. Специфікація OpenMP прийнята багатьма і підтримується такими постачальниками, як Sun, Intel, IBM і SGI. (Посилання на веб-сайт OpenMP, де є документ з останньою версією специфікації OpenMP, знаходиться нижче у розділі Література [17].)

Модель OpenMP дозволяє паралельному програмуванню піднятися на наступний рівень, створюючи для програміста потоки виконання і керуючи ними. Все, що необхідно, це вставити відповідні прагми у вихідний код програми і потім скомпілювати програму компілятором, який підтримує OpenMP, з відповідним ключем. Компілятор інтерпретує прагми і параллелізует код. При використанні компіляторів, що не підтримують OpenMP, прагми OpenMP ігноруються без додаткових повідомлень.

Основні конструкції OpenMP - це директиви компілятора або прагма (директиви препроцесора) мови C/C++. Нижче наведено загальний вигляд директиви OpenMP прагма.

```
#Pragma omp конструкція [пропозиція [пропозиція] ...]
```

У мові Фортран директиви OpenMP є коментарями і починаються з комбінації символів "!$OMP". Нижче наведені приклади таких рядків у загальному вигляді

```
!$OMP PARALLEL
!$OMP END PARALLEL
!$OMP PARALLEL DO ...
```

Для звичайних послідовних програм директиви OpenMP не змінюють структуру і послідовність виконання операторів. Таким чином, звичайна послідовна програма зберігає свою працездатність. У цьому й полягає гнучкість розпаралелювання за допомогою OpenMP.

Директиви можна розділити на 3 категорії:

- визначення паралельної секції,
- розподіл роботи,
- синхронізація.

Кожна директива може мати кілька додаткових атрибутів - *клауз (clause)* . Окремо специфікуються клаузи для призначення класів змінних, які можуть бути атрибутами різних директив.

Більшість директив OpenMP застосовується до структурних блоків. Структурні блоки - це послідовності операторів з однією точкою входу на початку блоку і однією точкою виходу в кінці блоку.

У OpenMP використовується модель паралельного виконання "розгалуження-злиття". Програма OpenMP починається як єдиний потік виконання (так званий початковий потік). Коли потік зустрічає паралельну конструкцію, він створює нову групу потоків, що складається з себе і невід'ємного числа додаткових потоків, і стає головним у новій групі. Всі члени нової групи (включаючи головний потік) виконують код всередині паралельної конструкції. У кінці паралельної конструкції є неявний бар'єр. Після паралельної конструкції виконання користувацького коду продовжує тільки головний потік.

Кількість потоків у групі, що виконуються в області паралельності, можна контролювати декількома способами. Один з них - використання змінної середовища `OMP_NUM_THREADS`. Інший спосіб - виклик процедури `omp_set_num_threads()`. Ще один спосіб - використання клаузи `num_threads` у поєднанні з прагмою `parallel`.

У OpenMP підтримуються дві основних конструкції розділення праці для зазначення того, що роботу в області паралельності слід розділити між потоками групи. Ці конструкції розділення праці - цикли і розділи. прагма `#pragma omp for` використовується для циклів, а прагма `#pragma omp sections`

використовується для розділів - блоків коду, які можуть бути виконані паралельно.

Прагма `#pragma omp barrier` дає всіх потоків вказівку чекати один одного перед тим, як вони продовжать виконання за бар'єром. Як було зазначено вище, в кінці області паралельності є неявний бар'єр. Прагма `#pragma omp master` дає компілятору вказівку про те, що наступний блок коду повинен виконуватися тільки головним потоком. Прагма `#pragma omp single` показує, що наступний блок коду повинен виконуватися тільки одним потоком групи; цей потік не обов'язково повинен бути головним. Прагма `#pragma omp critical` може використовуватися для захисту блоку коду, який повинен виконуватися одночасно тільки одним потоком. Звичайно, всі ці прагма мають сенс тільки в контексті прагма parallel (області паралельності).

7.3.1 Породження потоків виконання

`PARALLEL ... END PARALLEL`

Визначає *паралельну область* програми. При вході в цю область породжуються нові (N-1) потоки виконання, утворюється "команда" з N потоків, а головний процес отримує номер 0 і стає основним потоком команди (так званим "master thread"). При виході з паралельної області основний потік виконання чекає завершення інших потоків, і продовжує виконання в єдиному екземплярі. Передбачається, що в SMP-системі потоки будуть розподілені по різних процесорам. Однак це, як правило, перебуває у віданні операційної системи.

Яким чином між породженими потоками **розподіляється робота** - визначається директивами `DO`, `SECTIONS` і `SINGLE`. Можливо також явна управління розподілом роботи (*подібно MPI*) за допомогою функцій, що повертають номер даного потоку і загальне число потоків. За замовчуванням (за цими директивами), код всередині `PARALLEL` виконується всіма потоками однаково.

Разом з `PARALLEL` може використовуватися клауза `IF (умова)`. При цьому паралельна робота ініціюється тільки при виконанні зазначеного в ній умови.

Паралельні області можуть динамічно вкладеними. За замовчуванням (якщо вкладений паралелізм не дозволений явно), внутрішня паралельна область виконується одним потоком.

7.3.2 Розподіл роботи (work-sharing constructs)

Паралельні цикли

`DO ... [ENDDO]`

Визначає паралельний цикл.

Клауза `SCHEDULE` визначає спосіб розподілу ітерацій по нитках:

- `STATIC,m` - статично, блоками по m ітерацій

- `DYNAMIC,m` - динамічно, блоками по `m` (кожна нитка бере на виконання перший ще нездоланих вершин блок ітерацій)
- `GUIDED,m` - розмір блоку ітерацій зменшується експоненціально до величини `m`
- `RUNTIME` - вибирається під час виконання.

За замовчуванням, в кінці циклу відбувається неявна синхронізація; цю синхронізацію можна заборонити з допомогою клаузи `ENDDO NOWAIT`.

Паралельні секції

`SECTIONS ... END SECTIONS`

Не-ітеративна паралельна конструкція. Визначає набір незалежних секцій коду (так званий "кінцевий" паралелізм). Секції відокремлюються один від одного директивою `SECTIONS`.

Примітка. Якщо всередині PARALLEL міститься тільки одна конструкція `DO` або тільки одна конструкція `SECTIONS`, то можна використовувати скорочений запис: `PARALLEL DO` або `PARALLEL`.

Виконання одним потоком

`SINGLE ... END SINGLE`

Визначає блок коду, який буде виконано тільки одним потоком (першим, що дійде до цього блоку).

Явне управління розподілом роботи

За допомогою функцій `OMP_GET_THREAD_NUM()` і `OMP_GET_NUM_THREADS()` потік виконання може взнати свій номер і загальну кількість потоків, а потім виконувати свою частину роботи залежно від свого номера. (Цей підхід широко використовується в програмах на базі інтерфейсу MPI).

7.3.3 Директиви синхронізації

`MASTER ... END MASTER`

Визначає блок коду, який буде виконанувати тільки master (нульовий потік).

`CRITICAL ... END CRITICAL`

Визначає критичну секцію, тобто блок коду, який не повинен виконуватися одночасно двома або більше потоками.

`BARRIER`

Визначає точку бар'єрної синхронізації, в якій кожен потік чекає всіх інших.

`ATOMIC`

Визначає змінну в лівій частині оператора "атомарного" присвоювання, яка повинна коректно оновлюватися декількома потоками.

ORDERED ... END ORDERED

Визначає блок усередині тіла циклу, який повинен виконуватися в тому порядку, в якому ітерації йдуть у послідовному циклі. Може використовуватися для впорядкування виведення від паралельних потоків виконання.

FLUSH

Явно визначає точку, в якій реалізація повинна забезпечити однаковий зміст пам'яті для всіх потоків. Неявно FLUSH присутній у наступних директивах: BARRIER, CRITICAL, END CRITICAL, END DO, END PARALLEL, END SECTIONS, END SINGLE, ORDERED, END ORDERED.

Для синхронізації можна також користуватися механізмом замків (locks).

7.3.4 Класи змінних

В OpenMP змінні в паралельних областях програми розділяються на два основні класи:

- SHARED (глобальні; під ім'ям А всі потоки бачать одну змінну)
- PRIVATE (приватні; під ім'ям А кожен потік бачить свою змінну).

Окремі правила визначають поведінку змінних при вході і виході з паралельної області або паралельного циклу: REDUCTION, FIRSTPRIVATE, LASTPRIVATE, COPYIN.

За замовчуванням, всі COMMON-блоки, а також змінні, породжені поза паралельною областю, при вході в цю область залишаються глобальними (SHARED). Виняток становлять змінні - лічильники ітерацій в циклі, з очевидних причин. Змінні, породжені всередині паралельної області, є приватними (PRIVATE). Явно призначити клас змінних за замовчуванням можна за допомогою клаузи DEFAULT.

SHARED

Застосовується до змінних, які необхідно зробити глобальними.

PRIVATE

Застосовується до змінних, які необхідно зробити приватними. При вході в паралельну область для кожного потоку виконання створюється окремий екземпляр змінної, який не має жодного зв'язку з оригінальною змінною поза паралельною областю.

THREADPRIVATE

Застосовується до COMMON-блоків, які необхідно зробити приватними. Директива повинна застосовуватися після кожної декларації COMMON-блоку.

FIRSTPRIVATE

Приватні копії змінної при вході в паралельну область ініціалізуються значенням оригінальної змінної.

LASTPRIVATE

Після закінчення паралельно циклу або блоку паралельних секцій, потік, який виконав останню ітерацію циклу або останню секцію блоку, оновлює значення оригінальної змінної.

`REDUCTION(+:A)`

Вказує на змінну, з якою в циклі здійснюється операція редукції (наприклад, в даному випадку підсумовування). При виході з циклу, дана операція проводиться над копіями змінної у всіх потоках, і результат присвоюється оригінальній змінній.

`COPYIN`

Застосовується до COMMON-блоків, які позначені як THREADPRIVATE. При вході в паралельну область приватні копії цих даних ініціалізуються оригінальними значеннями.