Использование Intel Thread Checker для отладки многонитевых программ

Александр Позднеев
Аспирант 2-го года кафедры АНИ
pozdneev@gmail.com

Факультет вычислительной математики и кибернетики
Московский государственный университет имени М. В. Ломоносова

Специальный курс «Терафлопные вычисления»

План

Необходимое ПО

Microsoft Windows

- Ярлык Intel Thread Checker на рабочем столе и/или в меню «Пуск» (Intel Software Development Tools → Intel(R) Thread Checker x.x)
- `icl` (в консольном окне Intel Software Development Tools → Intel C++ Compiler xx.x.xxx → C++ Build Environment for applications running on . . .)
- `nmake` (в консоли)
- Текстовый редактор (FAR, etc.)

GNU/Linux

- `icc, gcc`
- `tcheck_cl`
- `make`

Мотивация

Гейзенбаг (англ. Heisenbug) —

программная ошибка, которая — по аналогии с принципом неопределенности Гейзенберга в квантовой физике — исчезает или видоизменяется при попытке ее выявления

Основные проблемы, которые могут возникнуть при взаимодействии параллельных процессов

- Гонка за данными (data races)
  - Два и более процесса обращаются к памяти без синхронизации (маловероятно, что два обращения к переменной произойдут *в один и тот же момент времени*, но очень даже возможно, что *порядок*, в котором эти обращения будут происходить, окажется неопределенным)
- Взаимная блокировка (deadlocks)
  - Для дальнейшего выполнения каждого из процессов требуются ресурсы, захваченные другим

Intel Thread Checker

- Средство отладки многопоточных программ
- Предназначен для поиска ошибок распараллеливания в программах, использующих OpenMP-, POSIX- и Windows-нити
- Находит места в коде, поведение в которых может быть недетерминированным
- Может обнаружить следующие ошибки и недоработки:
  - Условия гонки (data races)
  - Взаимная блокировка (deadlocks)
  - «Замершие нити» (stalled threads)
    - ★ Нити, ждущие освобождения объектов синхронизации или других ресурсов
  - Потерянные сигналы
  - Abandoned locks
- Ошибке не обязательно происходить, чтобы Intel Thread Checker ее обнаружил

Этапы работы с Intel Thread Checker

1. Подготовить тестовый набор данных
   - Он должен покрывать весь код
   - Использовать как можно меньшие наборы тестовых данных
   - Лишние итерации лишь увеличат время исполнения
   - Intel Thread Checker *анализирует*, а не просто собирает статистику
   - Для OpenMP-циклов, как правило, достаточно трех итераций (первая, «средняя», последняя)
2. Подготовить программу к запуску в Intel Thread Checker
   - Использовать Intel-компилятор и отладку на уровне исходного кода
   - Использовать не-Intel-компилятор с отладочными ключами
   - Не перекомпилировать — использовать отладку на уровне бинарного кода
3. Запустить программу в Intel Thread Checker
4. Исправить ошибки
5. goto 2

Компиляция

Отладка на уровне исходного кода (source instrumentation)

- Включить поддержку Intel Thread Checker: `/Qtcheck` (`-tcheck`)
- Отключить оптимизацию: `/Od` (`-O0`)
- Генерировать отладочную информацию: `/debug` (`-g`)
- При компоновке (linkage) использовать многопоточные динамически подгружаемые библиотеки с отладочной информацией: `/MD`
- Задействовать base relocations: `/fixed:no`

Отладка на уровне бинарного кода (binary instrumentation)

- ITC вносит изменения в загруженную в память программу
- Если нет доступа к подходящему Intel-компилятору
- Если перекомпиляция займет много времени
- Если нет доступа к исходному коду

Демонстрация №1

Поиск и исправление ошибок в программе нахождения простых чисел

1. Скомпилировать командой `make` программу `prime`
   - `prime` находит количество простых чисел в указанном пользователем диапазоне методом «грубой силы»
2. Выполнить несколько запусков при
   - `set OMP_NUM_THREADS=1`
   - `set OMP_NUM_THREADS=2`
3. Обратить внимание, как меняются результаты
   - Есть ли ускорение?
   - Сколько простых чисел найдено?
4. Выполнить `make clean`, задействовать в `makefile` альтернативный набор флагов, перекомпилировать
5. Приступить к отладке с помощью Intel Thread Checker
   - Windows: см. далее
   - Linux: `user@domain:~> tcheck_cl ./prime`

Анализ результатов

Data races for glb_primes_found: diagnostics

ID	Short Description	Severity Name	Count	Context [Best]	Description	1st Access [Best]	2nd Access [Best]
1	Write -> Read data-race	Error	4 6	omp for	Memory read of glb_primes_found at "prime.c":104 conflicts with a prior memory write of glb_primes_found at "prime.c":104 (flow dependence)	"prime.c":104	"prime.c":104
2	Read -> Write data-race	Error	2 3	omp for	Memory write of glb_primes_found at "prime.c":104 conflicts with a prior memory read of glb_primes_found at "prime.c":104 (anti dependence)	"prime.c":104	"prime.c":104
3	Write -> Write data-race	Error	2 3	omp for	Memory write of glb_primes_found at "prime.c":104 conflicts with a prior memory write of glb_primes_found at "prime.c":104 (output dependence)	"prime.c":104	"prime.c":104

Классификация ситуаций гонки за данными

Гонки типа Read → Write

1st Access (1-й процесс)
S1: `privateA = sharedX`
2nd Access (2-й процесс)
S2: `sharedX = privateB`

Гонки типа Write → Read

1st Access (1-й процесс)
S1: `sharedX = privateA`
2nd Access (2-й процесс)
S2: `privateB = sharedX`

Гонки типа Write → Write

1st Access (1-й процесс)
S1: `sharedX = privateA`
2nd Access (2-й процесс)
S2: `sharedX = privateB`

Анализ результатов: продолжение

Data races for progress: diagnostics

ID	Short Description	Severity Name	C	Context	Description	1st Access [Best]	2nd Access [Best]
4	Read -> Write data-race	Error	4	omp for	Memory write of progress at "prime.c":70 conflicts with a prior memory read of progress at "prime.c":72 (anti dependence)	"prime.c":72	"prime.c":70

Анализ результатов: продолжение

Data races for last_percent_done: diagnostics

ID	Short Description	Severity Name	Count	Context [Best]	Description	1st Access [Best]	2nd Access [Best]
5	Write -> Read data-race	Error	4	omp for	Memory read of last_percent_done at "prime.c":74 conflicts with a prior memory write of last_percent_done at "prime.c":77 (flow dependence)	"prime.c":77	"prime.c":74

Анализ результатов: продолжение

Data races for last_percent_done: diagnostics

ID	Short Description	Severity Name	Count	Context [Best]	Description	1st Access [Best]	2nd Access [Best]
6	Read -> Write data-race	Error	4	omp for	Memory write of last_percent_done at "prime.c":77 conflicts with a prior memory read of last_percent_done at "prime.c":77 (anti dependence)	"prime.c":77	"prime.c":77
7	Write -> Read data-race	Error	4	omp for	Memory read of last_percent_done at "prime.c":77 conflicts with a prior memory write of last_percent_done at "prime.c":77 (flow dependence)	"prime.c":77	"prime.c":77
8	Write -> Write data-race	Error	4	omp for	Memory write of last_percent_done at "prime.c":77 conflicts with a prior memory write of last_percent_done at "prime.c":77 (output dependence)	"prime.c":77	"prime.c":77

Принципы работы Intel Thread Checker

- В программу вставляются функции библиотеки ITC, чтобы отслеживать
  - обращения к памяти
  - операции синхронизации
  - создание нитей
- Диагностируется только тот код, который исполнялся
  - Например, если выполилась лишь одна итерация цикла, то зависимость между итерациями не сможет быть обнаружена
- Для каждой области памяти, к которой происходит обращение сохраняется
  - логическое время обращения
  - информация о нити, осуществившей обращение
- Анализ OpenMP-программ:
  - Thread-Count Independent (TCI) (`/Qtcheck`, `/Qopenmp`)
    - ★ `omp_get_thread_num()` не поддерживается, потому что анализ основан на исполнении последовательного кода
  - Thread-Count Dependent (TCD) (`/Qopenmp`, `/MD`)

Демонстрация №2
Обнаружение ситуации взаимной блокировки

1. Скомпилировать программу `deadlock`
   - Модельный пример, демонстрирующий бездарное использование критических секций, приводящее к взаимной блокировке
   - Источник: `$(ITC)/Samples/Deadlock`
2. Посмотреть, как меняются от запуска к запуску результаты при
   - `set OMP_NUM_THREADS=1`
   - `set OMP_NUM_THREADS=2`
3. Перекомпилировать для Intel Thread Checker
4. Выполнить несколько запусков программы в Intel Thread Checker
   - Меняется ли диагностика?
   - Будет ли гонка за переменную `globalY`?

Анализ результатов

Data races for globalX: diagnostics

ID	Short Description	Severity Name	Count	Context[Best]	Description	1st Access[Best]	2nd Access[Best]
4	Write -> Read data-race	Error	1	"deadlock.c":49	Memory read of globalX at "deadlock.c":60 conflicts with a prior memory write of globalX at "deadlock.c":39 (flow dependence)	"deadlock.c":39	"deadlock.c":60
5	Write -> Read data-race	Error	1	"deadlock.c":49	Memory read of globalY at "deadlock.c":60 conflicts with a prior memory write of globalY at "deadlock.c":41 (flow dependence)	"deadlock.c":41	"deadlock.c":60

Схема возникновения взаимной блокировки

Процессы 1 и 2 хотят захватить ресурсы А и Б

Процесс 1 начинает с А

1. Захватывает ресурс А
2.
3.
4. Ожидает освобождения А

Процесс 2 начинает с Б

1.
2. Захватывает ресурс Б
3. Ожидает освобождения А
4.

Взаимная блокировка

Причина возникновения взаимной блокировки

заключается в том, что процессы блокируют объекты синхронизации в различном порядке

Анализ результатов
Potential deadlock: diagnostics

ID	Short Description	Severity Name	Count	Context[Best]	Description	1st Access[Best]	2nd Access[Best]
1	A sync object was acquired in the wrong order	Caution	1	WholeProgram 1	A synchronization object "deadlock.c":28 was acquired in the wrong order at "deadlock.c":26	"deadlock.c":26	"deadlock.c":28
2	A sync object was acquired in the wrong order	Caution	1	WholeProgram 1	A synchronization object "deadlock.c":40 was acquired in the wrong order at "deadlock.c":38	"deadlock.c":38	"deadlock.c":40

Заключение

На занятии было рассмотрено

- область применения ITC
- идеология функционирования ITC
- стратегия подготовки тестовых наборов данных
- компиляция программ для анализа с помощью ITC

Продемонстрировано

- поиск и ликвидация гонки за данными
- обнаружение и ликвидация взаимной блокировки

Ссылки

Данное занятие подготовлено на основе

- материалов Intel Software College
- курса «Multi-core Programming for Academia», прослушанного автором в рамках «Multi-core Technological School» (октябрь 2007, МЭИ(ТУ), Москва, Россия)
- Intel VTune™ Performance Environment Help

Дополнительные источники

- `http://ru.wikipedia.org/wiki/Гейзенбаг`
- `http://ru.wikipedia.org/wiki/Deadlock`

Для заинтересовавшихся слушателей

- `$(ITC)/Samples/`
- `$(ITC)/Samples/CodeExamplesGuide.pdf`