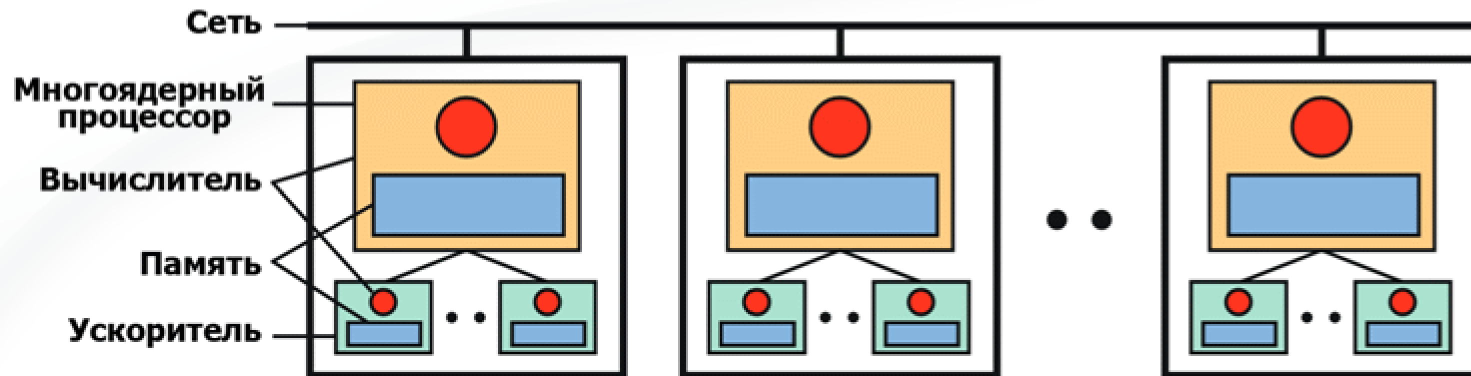
Отладка параллельных программ в DVM-системе

В.А. Бахтин, *Д.А. Захаров,*
А.А. Ермичев, В.А. Крюков

План доклада

1. DVM-система
2. Динамический контроль
3. Сравнительная отладка
4. Выводы

DVMH-модель параллелизма



- DVMH-модель позволяет создавать эффективные параллельные программы для гетерогенных вычислительных кластеров, в узлах которых в качестве вычислительных устройств (вычислителей) наряду с универсальными многоядерными процессорами могут использоваться ускорители различной архитектуры (графические процессоры, сопроцессоры Intel Xeon Phi, ...)
- При этом отображенные на узел вычисления автоматически распределяются между вычислителями узла с учетом их производительности

Система автоматизации разработки параллельных программ (DVM-система)

- Компиляторы с языков Fortran-DVMH и C-DVMH
  - преобразование входной программы в параллельную программу, использующую стандартные технологии программирования MPI, OpenMP и CUDA
- Библиотека поддержки LIB-DVMH
  - реализация модели выполнения параллельной программы;
  - динамическая настройка DVMH-программ на выделенные для их выполнения ресурсы: количество узлов кластера, ядер, ускорителей и их производительность
- DVMH-отладчик
  - автоматизированная отладка параллельных программ с использованием методов сравнительной отладки и динамического контроля корректности
- Анализатор производительности DVMH-программ
  - получение информации об основных характеристиках эффективности выполнения параллельной программы и ее частей на вычислительной системе

DVM-система – пример

```
double precision a(n, n), b(n, n)


<...>
do while ((it <= itmax).and.(eps > maxeps))
  eps = 0.D0


  do j = 2, n - 1
    do i = 2, n - 1
      b(i, j) = (a(i-1, j) + a(i+1, j) + a(i, j-1) + a(i, j+1)) / 4.D0
    enddo
  enddo


  do j = 2, n - 1
    do i = 2, n - 1
      eps = max(eps, abs(a(i, j) - b(i, j)))
      a(i, j) = b(i, j)
    enddo
  enddo

  it = it + 1
enddo
```

DVM-система – пример

```
      double precision a(n, n), b(n, n)
!dvm$ distribute(block, block) :: a
!dvm$ align b(i,j) with a(i, j)

      <...>
      do while ((it <= itmax).and.(eps > maxeps))
        eps = 0.D0
!dvm$   actual(eps)
!dvm$   region inout(a,b,eps)
!dvm$   parallel(j, i) on a(i, j), shadow_renew(a)
        do j = 2, n - 1
          do i = 2, n - 1
            b(i, j) = (a(i-1, j) + a(i+1, j) + a(i, j-1) + a(i, j+1)) / 4.D0
          enddo
        enddo

!dvm$   parallel(j, i) on a(i, j), reduction(max(eps))
        do j = 2, n - 1
          do i = 2, n - 1
            eps = max(eps, abs(a(i, j) - b(i, j)))
            a(i, j) = b(i, j)
          enddo
        enddo
!dvm$   end_region
        it = it + 1
      enddo
```

Динамический контроль в DVM-системе

Моделирование параллельного выполнения DVMH-программы на одном процессоре позволяет найти ошибки следующих типов:

- необъявленная зависимость по данным в параллельном цикле;
- использование в параллельном цикле или после выхода из него приватных переменных без их предварительной инициализации;
- запись в переменные, доступные только на чтение;
- использование редукционных переменных после запуска асинхронной редукции, но до ее завершения;
- необъявленный доступ к нелокальным элементам распределенного массива;
- запись в теневые грани массива;

Динамический контроль в DVM-системе

Моделирование параллельного выполнения DVMH-программы на одном процессоре позволяет найти ошибки следующих типов:

- чтение теневых элементов массива до завершения операции их обновления;
- модификация нелокального элемента распределенного массива в последовательной части программы;
- выход за пределы распределенного массива;
- запись в буфер удаленного доступа.

Сравнительная отладка в DVM-системе

- Результат выполнения эталонной программы записывается в файл трассы, а затем подается на вход отлаживаемому алгоритму.
  - Отсутствуют значительные временные затраты на синхронизацию параллельно работающих программ
  - Применение отладки ограничено большим объемом генерируемого файла трассы
- Автоматическая расстановка контрольных точек: трассируются все чтения и записи переменных, а также начало, конец и итерации циклов.
  - Есть возможность выбора событий для трассировки - только итерации циклов, только запись переменных, только обращения к распределенным массивам

Пример файла трассы

```
BW: [4] "eps"; {"test.f", 26}
AW: [4] "eps" = 0; {"test.f", 26}
PL: 2() [2]; {"test.f", 29}, 96.B
  IT: 18, (2,2)
  BW: [4] "b(i,j)"; {"test.f", 31}
  RD: [4] "a(i - 1,j)" = 681.72562479972839; {"test.f", 31}
  RD: [4] "a(i + 1,j)" = 551.343146201243б; {"test.f", 31}
  RD: [4] "a(i,j - 1)" = 681.72562479972839; {"test.f", 31}
  RD: [4] "a(i,j + 1)" = 551.343146201243б; {"test.f", 31}
  AW: [4] "b(i,j)" = 616.534385500486; {"test.f", 31}
  <...>
EL: 2; {"test.f", 33}, 96.E
PL: 3() [2]; {"test.f", 36}, 97.B
  IT: 18, (2,2)
  RV_BW: [4] "eps"; {"test.f", 38}
  RD: [4] "a(i,j)" = 616.51675929759654; {"test.f", 38}
  RD: [4] "b(i,j)" = 616.534385500486; {"test.f", 38}
  RV_AW: [4] "eps" = 0.017626202889459819; {"test.f", 38}
  BW: [4] "a(i,j)"; {"test.f", 39}
  RD: [4] "b(i,j)" = 616.534385500486; {"test.f", 39}
  AW: [4] "a(i,j)" = 616.534385500486; {"test.f", 39}
  <...>
```

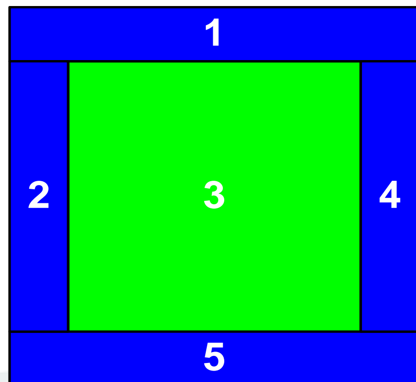
Проблемы сравнительной отладки

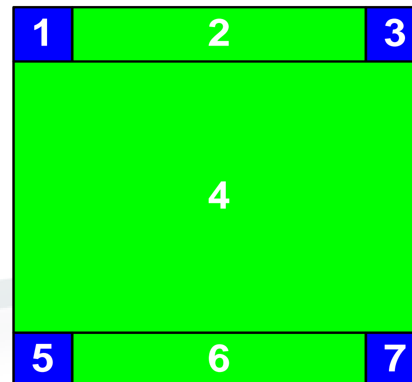
- **Ресурсы**: размер файла трассы может достигать нескольких гигабайт даже на небольших модельных задачах. Поэтому были реализованы режимы отладки, уменьшающие количество записей в трассу без заметного ущерба покрытию операторов отлаживаемой программы:
  - **Метод интегральных характеристик массива**, который позволял заменить трассировку обращений к элементам массива в параллельном цикле на подсчет и сравнение контрольной суммы значений массива после каждой итерации
  - **Метод граничных итераций**, трассирующий только граничные витки параллельных циклов, но при этом практически не теряющий точность определения ошибки.

Метод граничных итераций

- В циклах обработки массивов наиболее вероятным местом возникновения и проявления многих ошибок (например, использование неинициализированных переменных, выход за границы массива) являются граничные итерации
- Вычислительные алгоритмы часто таковы, что ошибка, возникшая на внутренней итерации, проявляется на граничной

грани





уголки

Метод граничных итераций для тестов NAS NPB класса A

	полн. инстр.	«уголки» ширины 1	«уголки» ширины 2	«грани» ширины 1	«грани» ширины 2
ср. покрытие опер., %	**100%**	**99,4%**	**99,8%**	**99,8%**	**99,8%**
ср. размер трасс, байт	**6,6E+12 (~6 Tb)**	**4,6E+07 (~44 Mb)**	**9,4E+08 (~894 Mb)**	**1,7E+10 (~16 Gb)**	**6,0E+10 (~56 Gb)**
ср. время генерации трасс, сек.	**27915 с. (7,75 ч.)**	**15**	**19**	**105**	**287**

Метод граничных итераций

Достоинства:

- существенное сокращение размеров трасс (сотни - сотни тысяч раз);
- существенное сокращение времен выполнения программ с генерацией трасс (десятки - тысячи раз);
- значительное сохранение покрытия операторов программы (более 99%).

Недостатки:

- возможен пропуск ошибок, если они не проявились на граничных итерациях, либо выявление при сравнительной отладке далеких последствий этих ошибок;
- необходимо учитывать возможность ложных диагностик об ошибках, например, «чтение неинициализированных данных».

Оценка объема трассы, получение конфигурационного «файла циклов»

```
SL: 2() [1] {"test.f", 24} =  #
  # Trace size = 6158756096 bytes
  # String count = 149400702
  # Count of traced iterations = 10
  # Used memory = 28684934864 bytes
  PL: 3(2) [2] {"test.f", 29} =  # , (0:2,999,1), (1:2,999,1)
    # Trace size = 3203528573 bytes
    # String count = 79680340
    # Count of traced iterations = 9960040
    # Used memory = 15378305600 bytes
  EL: 3
  PL: 4(2) [2] {"test.f", 33} =  # , (0:2,999,1), (1:2,999,1)
    # Trace size = 2955225361 bytes
    # String count = 69720300
    # Count of traced iterations = 9960040
    # Used memory = 13465977920 bytes
  EL: 4
EL: 2
<...>
```

Проблемы сравнительной отладки

- **Точность вещественных вычислений**: любая вещественная операция является потенциальным источником ошибок
- Результаты вещественных операций умножения и деления могут не совпасть в 1-2 младших знаках мантиссы при запуске:
  - на разных машинах
  - используя различные компиляторы
  - используя различные опции одного компилятора
- Изначально расхождения игнорируются отладчиком, но они:
  - накапливаются, вплоть до существенных для отладки значений
  - распространяются на другие переменные, участвующие в вычислениях
  - могут привести к расхождению потока управления двух программ

Развитие метода сравнительной отладки

Режим коррекции вещественных вычислений: замена результата вещественной операции на соответствующее значение из трассы

- Текущая реализация отладчика уже поддерживает похожий механизм для коррекции редукционных переменных
- Исключает применение оптимизаций размера трассы - необходима трассировка всех вещественных операций

Развитие метода сравнительной отладки

Реализация второго основного подхода к отладке: обмена отладочной информацией между двумя параллельно работающими программами

- Запуск двух инструментированных для отладки программ, синхронизация и обмен блоками трассы по сети
    - Вариант для кластеров - запуск двух вариантов программы в рамках одного пространства процессов *(например коммуникатор MPI)*
- Параллельные циклы - границы сравниваемых блоков трассы

Развитие метода сравнительной отладки

Реализация второго основного подхода к отладке: обмена отладочной информацией между двумя параллельно работающими программами

- Время работы отладчика увеличится:
  - Необходимо синхронизировать работу параллельно запущенных программ
  - Продолжение вычислений возможно только после обработки полученного блока трассы
- Затраты на машинную память: только для текущего блока
  - Для алгоритмов итерационных преобразований массивов затраты на хранение блока трассы будут пропорциональны размерам массивов
- Сохраняется применимость существующих оптимизаций размера трассы

Новые возможности сравнительной отладки

Специальный режим работы DVMH-программы, при котором все вычисления в регионах одновременно выполняются на процессоре и ускорителе

Сравнение данных осуществляется при входе и выходе из региона

Целочисленные данные сравниваются на совпадение, а вещественные числа сравниваются с заданной точностью по абсолютной и относительной погрешности

В случае нахождения расхождений выдается информация о найденных расхождениях

Далее в программе используется та версия данных, которая была получена при выполнении на центральном процессоре

Новые возможности сравнительной отладки. Обнаруживаемые ошибки

- Программистом произведено некорректное распараллеливание, не подходящее для массивно-параллельного выполнения в общей памяти
- Программист некорректно указал приватные или редукционные переменные в параллельном цикле
- Арифметические операции или математические функции на ускорителе отработали с иным по сравнению с работой на центральном процессоре результатом. Это может происходить из-за различий в системе команд, приводящих к различным результатам (в пределах точности округлений)
- Программист указал неверные директивы актуализации данных **get_actual** и **actual**, вследствие чего обрабатываемые данные на центральном процессоре и ускорителе оказались разными.

Экспериментальная версия системы сравнительной отладки

Осуществляет сравнение параллельно работающих программ

“Быстрая” отладка: сравнение на границе блоков значений всех переменных, задействованных в вычислениях внутри текущего блока

- Точность определения ошибки ограничена блоком, в котором она проявилась

- Затраты по времени и памяти на порядки меньше существующих
  - Не требуется обработка методами отладчика результата каждой операции чтения/записи переменной
  - При использовании совместно с методом интегральных характеристик массивов объем данных, требующих сравнения, на порядки меньше количества операций в блоке

Результаты экспериментальных запусков

	N = 10 100 итераций		N = 100 100 итераций		N = 1000 1000 итераций	
	Размер трассы	*Время работы*	**Размер трассы**	*Время работы*	**Размер трассы**	*Время работы*
Без отладки	**-**	*0.42 s*	**-**	*0.45 s*	**-**	*3.02 s*
Существующая реализация	**4.6 Mb**	*0.5 s*	**694 Mb**	*9.35 s*	**~800 Gb**	*~12000 s*
Сравнение одновременно выполняющихся программ	**1.08 Mb** *(all)* **7.4 Kb** *(block)*	*0.92 s*	**163 Mb** *(all)* **1.1 Mb** *(block)*	*31.2 s*	**~180 Gb** *(all)* **113 Mb** *(block)*	*~18000 s*

- Время отладки с коррекцией вещественных вычислений практически не отличается от времени выполнения полной отладки без оптимизаций

Выводы

- DVM-система автоматизирует процесс разработки параллельных программ для кластеров, использующих многоядерные универсальные процессоры, графические ускорители и сопроцессоры Intel Xeon Phi
- Важным преимуществом DVM-системы является наличие мощных инструментов для отладки получаемых в процессе распараллеливания программ. Данные инструменты используют метод динамического контроля и метод сравнительной отладки
- Разрабатывается новая версия системы сравнительной отладки, в которой и эталонная, и отлаживаемая программа будут выполняются одновременно, и сравнение результатов выполнения осуществляться «на лету» без трасс. Создаваемая версия системы решит проблему точности сравнения, сделает процесс отладки более гибким и менее требовательным к памяти вычислительного комплекса

Спасибо за внимание

DVM-СИСТЕМА

http://dvm-system.org

dvm@keldysh.ru