Новосибирский государственный университет
Факультет информационных технологий
Кафедра параллельных вычислений

Средство анализа причин зависаний фрагментированных программ в системе LuNA

Выполнил:
студент группы 18201 Мичуров Михаил Антонович
Научный руководитель:
Власенко Андрей Юрьевич, к.т.н., доц. каф. ПВ ФИТ НГУ

Описание темы

Данный проект направлен на автоматизированное решение проблемы обнаружения причин зависания фрагментированных программ.

В рамках системы фрагментированного программирования LuNA разрабатывается модуль обратной трассировки, выполняющий обнаружение неподготовленных входных данных для "зависшего" фрагмента вычислений.

Система LuNA и проблема зависаний

Система LuNA - это инструмент для построения параллельных программ на базе технологии фрагментированного программирования. LuNA состоит из транслятора языка сборки и подсистемы исполнения фрагментированных программ (runtime-системы).

Ошибки, приводящие к зависаниям runtime-системы, включают:

- Ошибки во внешних процедурах (C/C++);
- Семантические ошибки в коде LuNA-программы:
  - Циклические зависимости по данным;
  - Использование неинициализированных данных;
  - Ошибки при индексации.

Ошибка при индексации – пример

```
sub main() {
  // ...
  for i=0..N-1 {
    for j=0..N-1 {
      init_mat(0, i*M, j*M, M, M, A[i][j]);
      init_mat(1, i*M, j*M, M, M, B[i][j]);
      calc_mat(A, B, C[i][j], i, j, N);
    }
  }
}

sub calc_mat(name A, name B, name C, int i, int j, int N) {
  // ...
  for k=0..N {
    mult_mat(A[i][k], B[k][j], Ctmp[k]);
  }
  // ...
}
```

Фрагменты вычислений и их выполнение

Фрагмент вычислений (ФВ) – независимая единица программы, которая содержит описание входных и выходных фрагментов данных, а также код фрагмента. ФВ разделяют на атомарные и структурированные.

Выполнение фрагментов вычислений

Понятие зависания (1)

Под зависшим фрагментом вычислений понимается такой ФВ, который по тем или иным причинам не может завершиться ни при каких обстоятельствах в рамках штатной работы runtime-системы LuNA.

Фрагмент вычислений может зависнуть

- На этапе подготовки
  - Бесконечное ожидание данных, которые не создает ни один ФВ
- На этапе исполнения
  - Внешняя процедура может зависнуть из-за внутренних алгоритмических ошибок

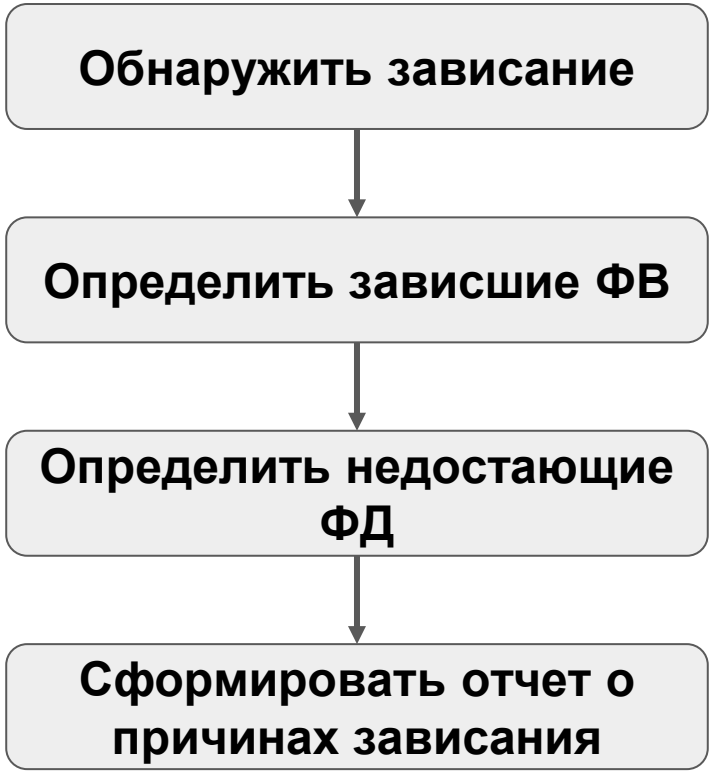
Понятие зависания (2)

Зависание одного или нескольких ФВ приводит к невозможности завершения работы runtime-системы.

Поиск причин зависания является нетривиальной задачей, поскольку:

- Ожидание данных является нормальным поведением.
- Высокий уровень абстракции при написании фрагментированного алгоритма затрудняет отладку программы во время исполнения.

Анализ причин зависания



Идея анализа причин зависания

Сбор отладочной информации

Runtime-система LuNA была расширена модулем, отвечающим за сбор отладочной информации во время выполнения программы.

Отладочная информация включает:

- Список ФВ, выполнение которых началось, но не завершилось
  - Сами ФВ также хранят информацию о входных и выходных ФД, об ожидаемых ФД
- Соответствие идентификаторов фрагментов данных имени и месту объявления ФД

При остановке работы runtime-системы вследствие получения SIGINT (Ctrl+C) собранная информация сохраняется на диск в лог-файлы, которые позже можно проанализировать с помощью созданной в рамках работы утилиты luna_trace.

Лог-файлы

Каждый узел сохраняет отладочную информацию в два лог-файла – *pending.<номер узла>.json* и *id.<номер узла>.map*.

id.<номер узла>.map*:*

- Хранит информацию о соответствии идентификатора ФД во время исполнения и его имени и места создания.

pending.<номер узла>.json:

- Хранит список зависших ФВ и для каждого такого ФВ хранит
  - Идентификаторы входных и выходных ФД;
  - Информацию об ожидаемых ФД;
  - Номер блока;
  - Информацию о родительских ФВ.

Обработка лог-файлов: luna_trace

Утилита командной строки luna_trace используется для анализа сгенерированных лог-файлов.

Функционал luna_trace включает:

- Вывод информации о ФД, отсутствие которых стало причиной зависания
  - Имя (локальное/использованное при объявлении)
  - Место объявления
  - Группировка ФД с индексированными именами
- Обнаружение “первопричин” зависания
  - Опционально – вывод информации обо всех зависших ФВ без поиска первопричин
- Вывод истории вызовов для ФВ

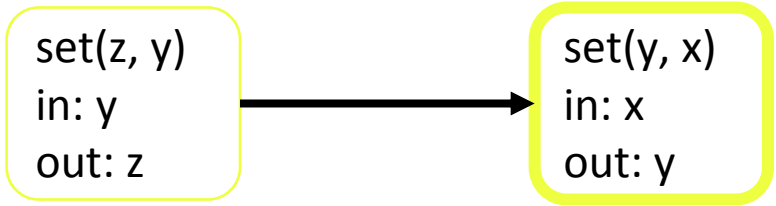
“Первопричины” зависаний и их поиск

Первопричиной зависания считается ФВ, не имеющий зависимостей по данным от других зависших ФВ.

При поиске первопричин строится ориентированный граф информационных зависимостей между зависшими ФВ на основании информации о входных и выходных ФД, сохраненной в лог-файлах.

```
C++ sub set(name x, int v) ${<!} {
  x.setValue<int>(v);
}!

sub main() {
  df x, y, z;
  set(y, x);
  set(z, y);
}
```



Граф информационных зависимостей для программы слева

Сценарий работы со средством анализа зависаний

1. Пользователь пишет код на C/C++ и входном языке LuNA как обычно
2. Программа собирается в отладочном режиме без удаления промежуточных файлов
3. Программа выполняется
4. Пользователь обнаруживает зависание программы и завершает ее работу с помощью Ctrl+C (отправка SIGINT)
   - Генерируются лог-файлы
5. luna_trace используется для получения отчета о причинах зависания

Примеры вывода luna_trace (1): сокращенный вывод диапазона имён

```
C++ sub set(name x, int v) ${<!} {
  x.setValue<int>(v);
}!

C++ sub print(int v) ${<!} {
  printf("%d\n", v);
}!

sub main() {
  df x;
  set(x[2500], 0);
  for i = 1..2500 {
    print(x[i * 2]);
  }
}
```

```
Following CFs appear to be the root cause of the system hang:

print(x[i * 2]) [./many.fa:13] never finished (2499 instances)
main -> print
  in for i = 1..2500 [./many.fa:12]
  in sub main() [./many.fa:9]

  awaited DF x[(i)*(2)] (where x is x, x declared in sub main)
    step = 2
    x[2]..x[2498] (x[2]..x[2498])
    x[2502]..x[5000] (x[2502]..x[5000])
```

Примеры вывода luna_trace (2): первопричина зависания

```
C++ sub set(name x, int v) ${<!} {
  x.setValue<int>(v);
}!

C++ sub print(int v) ${<!} {
  printf("%d\n", v);
}!

sub main() {
  df a, b, c;
  set(b, a * a);
  set(c, b + 1);
  print(c);
}
```

```
Following CFs appear to be the root cause of the system hang:

set(b, a * a) [./root.fa:11] never finished
main -> set
  in sub main() [./root.fa:9]

  awaited DF a (a declared in sub main)

- - - -

2 more CFs never finished but had data dependencies on other hang CFs
```

Примеры вывода luna_trace (3): циклические зависимости

```
C++ sub set(name x, int v) ${<!} {
  x.setValue<int>(v);
}!

sub main() {
  df x, y, z;
  set(y, x);
  set(z, y);
  set(x, z);
}
```

```
Cannot find a single root cause: hang CFs have cyclic dependencies!

set(y, x) [./cyclic.fa:7] never finished
main -> set
  in sub main() [./cyclic.fa:5]

  awaited DF x (x declared in sub main)

- - - -

set(z, y) [./cyclic.fa:8] never finished
main -> set
  in sub main() [./cyclic.fa:5]

  awaited DF y (y declared in sub main)

- - - -

set(x, z) [./cyclic.fa:9] never finished
main -> set
  in sub main() [./cyclic.fa:5]

  awaited DF z (z declared in sub main)
```

Оценка накладных расходов (1)

Была произведена оценка накладных расходов при сборе отладочной информации на нескольких тестовых программах с варьируемыми параметрами. Оценивалось замедление программы и увеличение пикового использования памяти.

- Измерения проводились с помощью утилиты time.
- Число рабочих потоков в runtime-системе: 4
- Процессор: Intel(R) Core(TM) i7-3537U CPU @ 2.00GHz
- Число ядер: 2
- Число логических процессоров: 4
- RAM: 12,0 ГБ DDR3
- Число MPI-процессов: 1

Оценка накладных расходов (1): блочное умножение матриц

Размер блока	**8**	**8**	**8**	**8**	**8**	**20**	**20**	**20**	**20**	**20**
Число блоков	65	70	75	80	85	26	28	30	32	34
Замедление, %	35	28	35	35	**40**	**17**	25	21	28	22
Увеличение использования памяти, %	21	19	19	**23**	18	6	6	**5**	5	7

Оценка накладных расходов (2): вычисление числа пи через частичную сумму ряда

Число членов ряда	3`000`000	4`000`000	5`000`000
Замедление, %	30	**44**	34
Увеличение использования памяти, %	**48**	41	42

Дальнейшая работа

В дальнейшем планируется тестирование разработанного программного средства на большем числе LuNA-программ, вызывающих зависание. Также планируется оптимизировать процесс сбора отладочной информации с целью снижения накладных расходов и автоматизировать обнаружение зависаний runtime-системы.

Спасибо за внимание!