Использование динамического анализа при распараллеливании на гибридный вычислительный кластер

Н.А. Катаев, А.А. Смирнов

Институт прикладной математики им. М.В. Келдыша РАН

29 ноября, 2022 | Переславль-Залесский

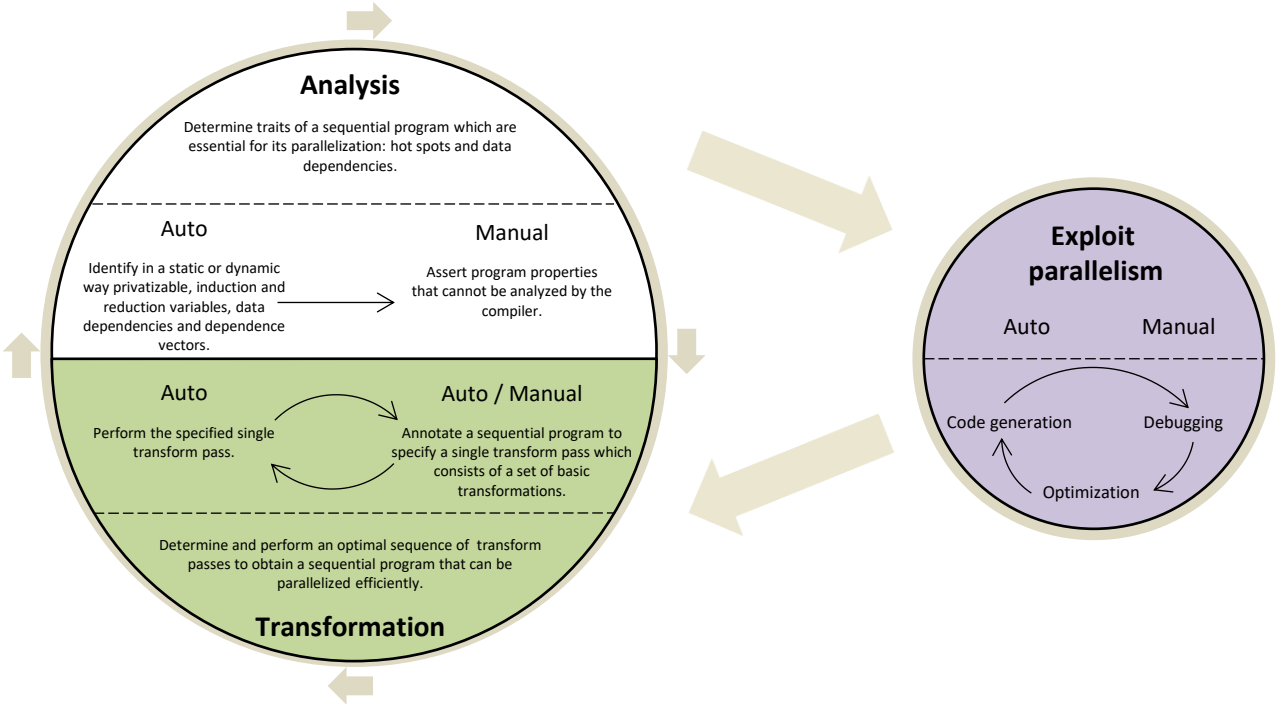
Инструменты параллельного программирования

Производительность		Удобство
MPI, CUDA, OpenCL, SHMEM, pThreads	OpenMP, OpenACC, DVMH, XcalableACC, Halid, Vobla, Graphit, MKL, Thrust, cuBLAS	SAPFOR, PPCG, Polly, Pluto, Apollo, Paradigm, SUPERB
Низкоуровневые модели дают программисту полный контроль над выполнением программы и позволяют ему добиться наилучшей производительности.	Директивные модели, DSLs, библиотеки общего назначения упрощают программирование и повышают удобство сопровождения ПО, обеспечивая при этом высокую производительность.	Автоматически распараллеливающие компиляторы создают параллельный код для входной программы (но не всегда оптимальный).

Смешанный подход к параллельному программированию

Максимальная автоматизация процесса распараллеливания, при активном участии пользователя.

- Процесс распараллеливания рассматривается как последовательность отдельных этапов.
- Каждый этап может быть автоматизирован, если это возможно, или выполнены вручную.



DVMH-модель

Язык параллельного программирования высокого уровня.
Модель программирования на основе директив.

DVMH

CUDA

OpenMP

MPI

Модель программирования на основе директив, которая направлена на создание параллельных программ для гетерогенных вычислительных кластеров (GPU NVidia, Intel Xeon Phi, многоядерные процессоры).

Модель включает в себя два языка программирования, которые являются расширениями стандартных языков C и Fortran спецификациями параллелизма: CDVMH и Fortran-DVMH

Параллельная программа разрабатывается в терминах последовательной.

Особенности DVMH модели

Описание параллелизма

- Параллельная программа разрабатывается в терминах последовательной.
- Возможно как последовательное так и параллельное выполнение программы благодаря тому, что обычный компилятор игнорирует спецификации параллелизма.
- В коде программы отсутствуют низкоуровневые спецификации обмена данными и синхронизации.
- Высокий уровень спецификаций параллелизма обеспечивает удобство разработки и сопровождения программы.

Методы динамической настройки программы

- Система поддержки контролирует выполнение программы и настраивает ее на все доступные вычислительные ресурсы.

- Оптимизации невидимы для пользователя:
  - ✓ оптимизация расположения данных в памяти в зависимости от особенностей вычислительного устройства,
  - ✓ динамическая компиляция CUDA обработчиков,
  - ✓ параллельное выполнение циклов с зависимостями по данным.

Функциональная отладка (динамический контроль, сравнительная отладка) и отладка производительности

- Высокоуровневое описание найденных проблем в терминах конструкций модели программирования.

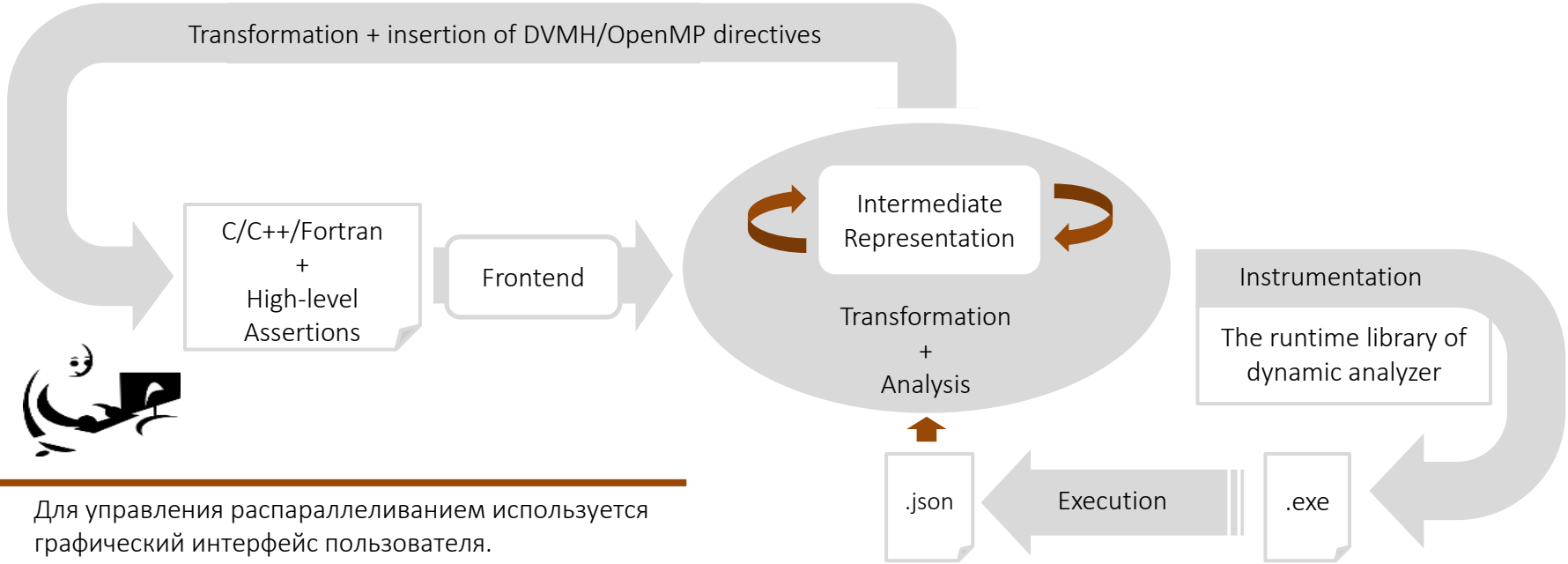
Система SAPFOR

SAPFOR (System For Automate Parallelization) является системой для разработки параллельных программ, ориентированной на снижение затрат на ручное распараллеливание программ.

Главные цели разработки системы SAPFOR:

- Исследование последовательных программ (анализ и профилирование программ).
- Автоматическое распараллеливание (в соответствии с моделями DVMH или OpenMP) потенциально параллельной программы, для которой программист максимизирует параллелизм на уровне алгоритма и/или добавляет высокоуровневые спецкомментарии свойств программы.
- Полуавтоматическое преобразование программы для получения потенциальной последовательной версии исходной программы.

Архитектура системы SAPFOR



- Для управления распараллеливанием используется графический интерфейс пользователя.
- Инструменты автоматизации сборки, такие как Make, также можно использовать для выполнения анализа программ.

Отображение вычислений на GPU

В исходный код должны быть вставлены следующие виды директив:

- спецификации для циклов, которые могут выполняться параллельно (***parallel***), спецификации приватных (***private***) и редукционных (***reduction***) переменных и регулярных зависимостей по данным (***across***),
- спецификации вычислительных областей (***region***), которые могут быть выполнены на ускорителях, каждая область может содержать один или несколько параллельных циклов,
- высокоуровневые спецификации передачи данных (***actual, get_actual, in, out, local***) между памятью центрального процессора и памятью ускорителя (директивы актуализации данных).

```
#pragma dvm actual(...R...)
...
R = ...
...
#pragma dvm region in(R) out(W)
{
    #pragma dvm parallel([i]) \
                reduction(...) private(...)
    for (... i ...)
      W = R;
}
#pragma dvm get_actual(...W...)
...
... = W
...
```

Спецификация ***actual*** указывает начало динамической по графу управления области программы, в которой могут быть изменены данные во внерегионном пространстве, необходимые для выполнения последующих вычислительных регионов

Спецификация ***get_actual*** указывает точку в программе, в которой данные должны быть загружены с вычислительного устройства в оперативную память вычислительной системы

Автоматизация отображения вычислений на GPU

В исходный код должны быть вставлены следующие виды директив:

- спецификации для циклов, которые могут выполняться параллельно (***parallel***), спецификации приватных (***private***), редукционных (***reduction***) переменных и регулярных зависимостей по данным (***across***),
- спецификации вычислительных областей (***region***), которые могут быть выполнены на ускорителях, каждая область может содержать один или несколько параллельных циклов,
- высокоуровневые спецификации передачи данных (***actual, get_actual, in, out, local***) между памятью центрального процессора и памятью ускорителя (директивы актуализации данных).

```
#pragma dvm actual(...R...)
...
R = ...
...
#pragma dvm region in(R) out(W)
{
    #pragma dvm parallel([i]) \
                reduction(...) private(...)
    for (... i ...)
      W = R;
}
#pragma dvm get_actual(...W...)
...
... = W
...
```

Ограничения DVMH модели на код, который может быть отображен на GPU упрощают анализ внутрирегионного пространства.

Для каждого цикла рассматриваются следующие ограничения:

- безопасность потока управления (отсутствие операций ввода-вывода, побочных эффектов и т.д.),
- безопасность доступа к памяти (отсутствие зависимостей по данным в циклах и «захваченных» указателей),
- направление использования данных (входные, выходные и локальные данные),
- каноническая форма цикла в соответствии со стандартом OpenMP,
- возможность выразить свойство переменной с использованием спецификаций DVMH языков,
- возможность объединения итерационных пространств вложенных циклов в одно большее итерационное пространство.

Автоматизация отображения вычислений на GPU

В исходный код должны быть вставлены следующие виды директив:

- спецификации для циклов, которые могут выполняться параллельно (***parallel***), спецификации приватных (***private***), редукционных (***reduction***) переменных и регулярных зависимостей по данным (***across***),
- спецификации вычислительных областей (***region***), которые могут быть выполнены на ускорителях, каждая область может содержать один или несколько параллельных циклов,
- высокоуровневые спецификации передачи данных (***actual, get_actual, in, out, local***) между памятью центрального процессора и памятью ускорителя (директивы актуализации данных).

```
#pragma dvm actual(...R...)
...
R = ...
...
#pragma dvm region in(R) out(W)
{
    #pragma dvm parallel([i]) \
                reduction(...) private(...)
    for (... i ...)
      W = R;
}
#pragma dvm get_actual(...W...)
...
... = W
...
```

Анализ внерегионного пространства зависит от используемого режима DVM системы:

- в случае распределения данных в модели DVMH на вычислительный кластер становится невозможным неявное изменение данных, распределяемых между узлами кластера,
- в случае использования DVMH языков для дополнительного распределения MPI-программ ограничения на конструкции, используемые во внерегионном пространстве практически снимаются.

Расстановка спецификаций актуализации данных

Система поддержки времени выполнения DVM системы оптимизирует множественные указания одних и тех же диапазонов адресов в разных спецификациях actual/get_actual и выполняет посылку данных однократно при необходимости.

#pragma dvm actual(<array-range-list>)

- указывает начало динамической по графу управления области программы, в которой могут быть изменены данные во внерегионном пространстве, необходимые для выполнения последующих вычислительных регионов,

- необходимо ставить перед фактическим изменение данных во внерегионном пространстве, чтобы запретить временную выгрузку данных с устройства и не «испортить» значения переменных, вычисленных вне регионов.

#pragma dvm get_actual(<array-range-list>)

- указывает точку в программе, в которой данные должны быть загружены с вычислительного устройства в оперативную память вычислительной системы,

- необходимо располагать перед первым использованием во внерегионном пространстве данных, измененных в вычислительном регионе.

```
array-range ::= var-name [ subscript-range ]...
subscript-range ::= [ int-expr [ : int-expr ] ]
```

Проблемы автоматизированной расстановки спецификаций

Вызовы внешних функций и/или неявное изменение используемой в вычислительном регионе памяти не позволяют точно определить точку модификации и использования данных средствами статического анализа.

```
#pragma get_actual(<ВСЕ данные потенциально читаемые в foo>)
#pragma actual(<ВСЕ данные потенциально изменяемые в foo>)
foo(…);
```

Использование сложных составных конструкций языка (if-then-else, операторы циклов, тернарные операторы языка Си) усложняют расстановку спецификаций актуализации данных и могут делать ее невозможной без предварительного изменения исходного кода программы.

```
#pragma get_actual(R(expr1), R(expr2))
#pragma actual(W(expr1), W(expr2))
for (expr1; expr2; expr3) {
  <body>
  #pragma get_actual(R(expr2), R(expr3))
  #pragma actual(W(expr2), W(expr3))
}
```

$$!(region \in expr1) \;||\; (W(expr1) \cap R(expr2) == \emptyset$$

$$!(region \in expr3) \;||\; (W(expr3) \cap R(expr2) == \emptyset$$

Оптимизация передачи данных: группировка обменов

```
#define N 1000000

int S = 0;
for (int I = 0; I < N; ++I) {
#pragma dvm get_actual(A[I])
  S = S + A[I];
}
```

Proc: #1

GPU #1 (NVIDIA GeForce RTX 3050)

	#	Min	Max	Sum	Average	Productive	Lost
[Region IN] Copy CPU to GPU	1	3.815M	3.815M	3.815M	3.815M	0.0009s	-
GET_ACTUAL	1000000	4B	4B	3.815M	4B	76.6449s	-
Loop execution	1	0.0001	0.0001	0.0001	0.0001	0.0001s	-
Page lock host memory	2	0.0004	0.0017	0.0021	0.0010	-	0.0021s

Productive time: 76.6459s
Lost time : 0.0021s

Суммарно 3,815M передано за 76,6449s

```
#define N 1000000

int S = 0;
#pragma dvm get_actual(A)
for (int I = 0; I < N; ++I) {
  S = S + A[I];
}
```

Proc: #1

GPU #1 (NVIDIA GeForce RTX 3050)

	#	Min	Max	Sum	Average	Productive	Lost
[Region IN] Copy CPU to GPU	1	3.815M	3.815M	3.815M	3.815M	0.0010s	-
GET_ACTUAL	1	3.815M	3.815M	3.815M	3.815M	0.0007s	-
Loop execution	1	0.0001	0.0001	0.0001	0.0001	0.0001s	-
Page lock host memory	2	0.0004	0.0017	0.0021	0.0010	-	0.0021s

Productive time: 0.0018s
Lost time : 0.0021s

Суммарно 3,815M передано за 0,0007s

Поэлементная передача массива выполняется в 109 493 раза медленнее

Оптимизация передачи данных: избыточные обмены

```
#define N 1000000

long S = 0;
for (int J = 0; J < 1000; ++J) {
  #pragma dvm region
  {
    #pragma dvm parallel([I]) reduction(sum(S))
    for (int I = 0; I < N; ++I) {
      S = S + A[I];
    }
  }
  #pragma dvm get_actual(A)
  #pragma dvm actual(A)
  A[0] = A[0] + 1;
}
```

```
#define N 1000000

long S = 0;
for (int J = 0; J < 1000; ++J) {
  #pragma dvm region
  {
    #pragma dvm parallel([I]) reduction(sum(S))
    for (int I = 0; I < N; ++I) {
      S = S + A[I];
    }
  }
  #pragma dvm get_actual(A[0])
  #pragma dvm actual(A[0])
  A[0] = A[0] + 1;
}
```

Proc: #1

GPU #1 (NVIDIA GeForce RTX 3050)

	#	Min	Max	Sum	Average	Productive	Lost
[Region IN] Copy CPU to GPU	1000	3.815M	3.815M	3.725G	3.815M	0.6518s	–
GET_ACTUAL	1	3.815M	3.815M	3.815M	3.815M	0.0007s	–
Loop execution	1001	0.0001	0.0014	0.2972	0.0003	0.2972s	–
Reduction	1000	0.0001	0.0007	0.1783	0.0002	–	0.1783s
Page lock host memory	2	0.0004	0.0019	0.0023	0.0012	–	0.0023s

Productive time: 0.9497s
Lost time : 0.1806s

Суммарно 3,729G передано за 0,6525s

Proc: #1

GPU #1 (NVIDIA GeForce RTX 3050)

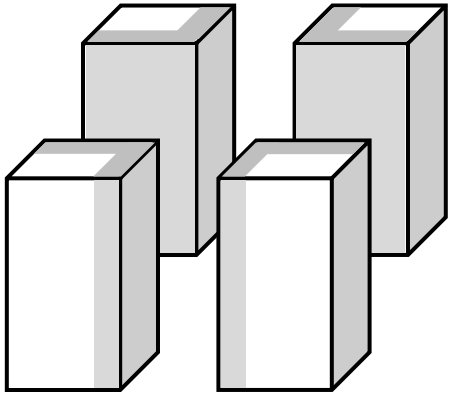
	#	Min	Max	Sum	Average	Productive	Lost
[Region IN] Copy CPU to GPU	1000	4B	3.815M	3.819M	3.909K	0.0472s	–
GET_ACTUAL	1	4B	4B	4B	4B	0.0000s	–
Loop execution	1001	0.0001	0.0014	0.2624	0.0003	0.2624s	–
Reduction	1000	0.0001	0.0006	0.1604	0.0002	–	0.1604s
Page lock host memory	2	0.0005	0.0018	0.0023	0.0011	–	0.0023s

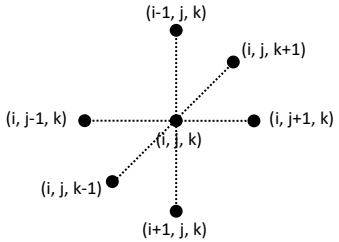
Productive time: 0.3097s
Lost time : 0.1627s

Суммарно 3,819M передано за 0.0472s

Избыточная передача всего массива выполняется в 14 раз медленнее

Пример: Himeno benchmark*





Размер данных:	513 x 513 x 1025
Количество итераций:	190
Количество блоков:	1 x 2 x 4
Размер блока:	513 x 259 x 259

Решение уравнения Пуассона методом Якоби.

```
253     for(i=1 ; i<imax-1 ; ++i)
254       for(j=1 ; j<jmax-1 ; ++j)
255         for(k=1 ; k<kmax-1 ; ++k){
256           s0 = a[0][i][j][k] * p[i+1][j  ][k  ]
257              + a[1][i][j][k] * p[i  ][j+1][k  ]
258              + a[2][i][j][k] * p[i  ][j  ][k+1]
259              + b[0][i][j][k] * ( p[i+1][j+1][k  ] - p[i+1][j-1][k  ]
260                                - p[i-1][j+1][k  ] + p[i-1][j-1][k  ] )
261              + b[1][i][j][k] * ( p[i  ][j+1][k+1] - p[i  ][j-1][k+1]
262                                - p[i  ][j+1][k-1] + p[i  ][j-1][k-1] )
263              + b[2][i][j][k] * ( p[i+1][j  ][k+1] - p[i-1][j  ][k+1]
264                                - p[i+1][j  ][k-1] + p[i-1][j  ][k-1] )
265              + c[0][i][j][k] * p[i-1][j  ][k  ]
266              + c[1][i][j][k] * p[i  ][j-1][k  ]
267              + c[2][i][j][k] * p[i  ][j  ][k-1]
268              + wrk1[i][j][k];
269
270           ss = ( s0 * a[3][i][j][k] - p[i][j][k] ) * bnd[i][j][k];
271           wgosa += ss*ss;
272
273           wrk2[i][j][k] = p[i][j][k] + omega * ss;
274         }
275
276     for(i=1 ; i<imax-1 ; ++i)
277       for(j=1 ; j<jmax-1 ; ++j)
278         for(k=1 ; k<kmax-1 ; ++k)
279           p[i][j][k] = wrk2[i][j][k];
280     sendp(ndx,ndy,ndz);
281     MPI_Allreduce(&wgosa, &gosa, 1, MPI_FLOAT, MPI_SUM, MPI_COMM_WORLD);
```

Конфигурация запуска	Copy (Gb)	Copy (s.)	Total (s.)
Исходная MPI версия (8 MPI)	N/A	N/A	62,51
MPI+DVMH (8 MPI + 8 GPU), *копирование между CPU и GPU всего массива p*	51,151	11,26	17,48
MPI+DVMH (8 MPI + 8 GPU), *копирования между CPU и GPU только теневых граней массива p*	2,557	1,41	6,42

K60 cluster (KIAM RAS):

1 node — 2 Intel Xeon Gold 6142v4 (16-core)
4 NVIDIA V100 GPU

*https://i.riken.jp/en/supercom/documents/himenobmt/

Динамический анализ

Цель – сократить объем передаваемых данных между CPU и GPU.

Задача – в исследуемой DVMH-программе выявить, какие данные в спецификациях actual/get_actual были указаны по необходимости, т.е. их отсутствие в директивах приведет к некорректному выполнению программы.

Предполагается, что имеющиеся DVMH директивы приводят к корректному, но, возможно, неэффективному выполнению программы.

Применение – рекомендовать пользователю в каких точках программы возможно выполнить оптимизацию спецификаций actual/get_actual за счет более точного указания диапазона адресов передаваемых объектов памяти.

Сформировать предполагаемый шаблон доступа в виде подмножества элементов многомерного массива:

```
#pragma dvm get_actual(p[0:imax][0:jmax][1], p[0:imax][0:jmax][kmax-2])
#pragma dvm actual(p[0:imax][0:jmax][0], p[0:imax][0:jmax][kmax-1])
```

Инструментация

Вставка в код программы вызовов внешней библиотеки

На уровне исходного кода

- Отдельная реализация для каждого поддерживаемого языка программирования.
- Ограниченные возможности использования оптимизаций компилятора.

На уровне внутреннего представления компилятора

- Единая реализация для разных языков программирования.
- Возможность выборочной инструментация в сочетании с выбором допустимых оптимизаций компилятора.

На уровне исполняемых файлов (бинарная инструментация)

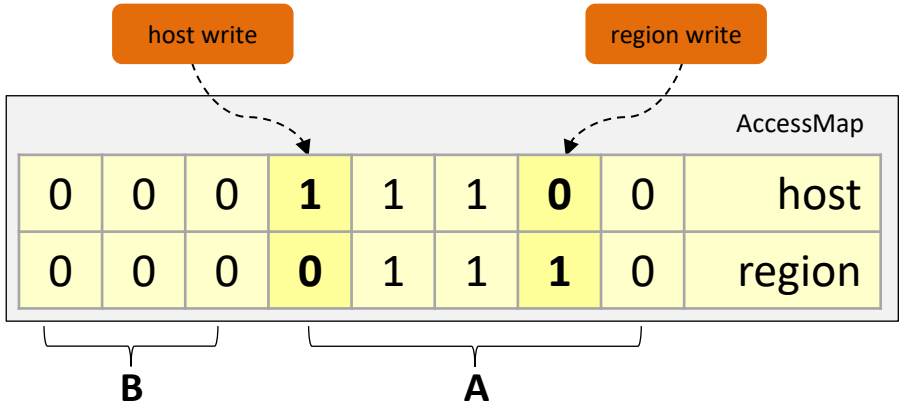
- Применима даже в случае отсутствия исходных кодов.
- Возможность соотнесения полученной информации с исходным кодом анализируемой программы сильно зависит от полноты доступной отладочной информации.
- Ограниченные возможности использования оптимизаций компилятора.

Инструментация и динамический анализ в SAPFOR

Программа представлена в виде LLVM IR.

Обрабатываются следующие объекты:

- операторы выделения и удаления памяти,
- обращения к памяти,
- вызовы функций,
- входы и выходы из регионов (region),
- спецификации передачи данных (actual/get_actual).



Каждый объект в коде программы идентифицируется контекстной строкой, содержащей описание объекта (имя, тип, позицию в исходном коде программы и т.д.).

Основные структуры данных динамического анализа:

- *глобальное хранилище результатов* накапливает результаты анализа на протяжении всей работы программы, а в конце на основе его содержимого строится выдача анализатора,
- *текущее состояние актуальности памяти* для каждого участка памяти, который встретился на данный момент выполнения программы, содержит информацию об актуальности переменной вне вычислительного региона и внутри вычислительного региона, а также точку первого изменения объекта вне региона.

Состояния алгоритма динамического анализа

STATE	cs_id	null	null	null	null	cs_0	cs_0	cs_0	cs_0
	HOST	0	0	1	1	0	0	1	1
	REGION	0	1	0	1	0	1	0	1
cs_1 WRITE	HOST	cs_1 1 0	cs_1 1 0		cs_1 1 0			ничего не делать	
	REGION	0 1	ничего не делать		0 1			0 1	
cs_2 READ	HOST	переменная без значения	1 1 добавить **get_actual** к cs_2		ничего не делать			ничего не делать	
	REGION	переменная без значения	ничего не делать		ничего не делать			1 1 добавить **actual** к cs_0	

Заключение

В системе SAPFOR реализован автоматически распараллеливающий компилятор, который подходит для распараллеливания потенциально параллельных программ без участия пользователя.

Пользователь может указывать свойства программы или определять последовательность необходимых преобразований.

Оптимизации, реализованные в компиляторе и в runtime системе DVMH, упрощают автоматизированную расстановку спецификация параллелизма, но не решают все проблемы, связанные с эффективностью выполнения полученной параллельной программы.

Средства динамического анализа применяются в системе SAPFOR для определения свойств программы, которые невозможно достоверно подтвердить статическим анализом.

Зависимость результатов анализа программы от используемой конфигурации запуска требует многократных запусков программ на разных наборах выходных данных и участия пользователя для подтверждения полученных результатов.

Интерактивная оболочка системы SAPFOR может быть использована для указания рекомендуемых для оптимизации точек в исходном коде программы с описанием возможных способов оптимизации.

Спасибо за внимание!

Site

http://dvm-system.org

https://github.com/dvm-system

GitHub

НСКФ
НАЦИОНАЛЬНЫЙ СУПЕРКОМПЬЮТЕРНЫЙ ФОРУМ