ВСЕРОССИЙСКАЯ НАУЧНАЯ КОНФЕРЕНЦИЯ

НАУЧНЫЙ СЕВИС В СЕТИ ИНТЕРНЕТ

Согласованное представление Fortran-программ в компиляторе PARFOR на разных уровнях абстракции

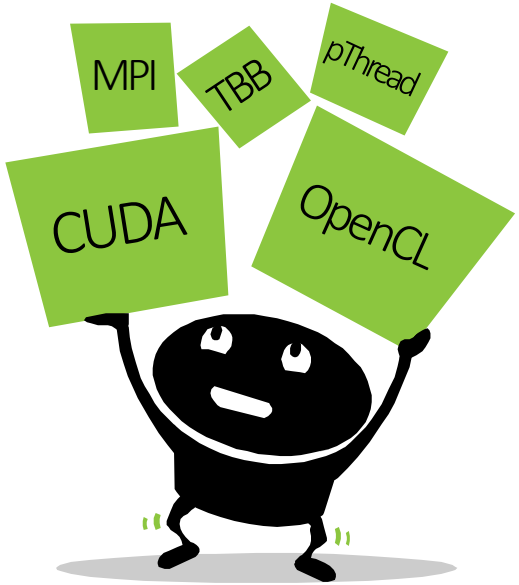
19 сентября 2018 г. | Абрау-Дюрсо

Н.А. Катаев

Институт прикладной математики им. М.В. Келдыша РАН

Распараллеливание – это сложно …

Имеющиеся средства, которые могут применяться для разработки прикладного программного обеспечения с использованием технологий параллельного программирования, требуют от пользователя детальных знаний используемых технологий и по своей сути являются низкоуровневыми.



Снижение сложности программирования

Разработка высокоуровневых языков параллельного программирования.

OpenMP

DVMH

hiCUDA

OpenACC

XcalableACC

Fortress

HPF

Chapel

X10

Снижение сложности программирования

Разработка высокоуровневых языков параллельного программирования.

Директивные расширения существующих языков программирования.

CUDA

OpenMP

MPI

DVMH

- Модель программирования, предназначенная для использования на кластерах с аппаратурой различной архитектуры (GPU NVidia, Intel Xeon Phi, multicore CPUs).
- Специальные комментарии и прагмы являются высокоуровневыми спецификациями параллелизма в терминах последовательной программы.
- Существует для двух языков: *C-DVMH* и *Fortran-DVMH*

Снижение сложности программирования

Разработка высокоуровневых языков параллельного программирования.

Директивные расширения существующих языков программирования.

CUDA

OpenM

MPI

```
#pragma dvm region
{
#pragma dvm parallel([i][j] on A[i][j]) reduction(max(eps))
for (i = 1; i < L - 1; i++)
  for (j = 1; j < M - 1; j++)
  {
    float tmp = fabs(B[i][j] - A[i][j]);
    eps = Max(tmp, eps);
    A[i][j] = B[i][j];
  }

#pragma dvm parallel([i][j] on A[i][j]) shadow_renew(A)
for (i = 1; i < L - 1; i++)
  for (j = 1; j < M - 1; j++)
      B[i][j] = (A[i-1][j] + A[i][j-1] + A[i][j+1] + A[i+1][j]) / 4.0f;
}
```

Снижение сложности программирования

Исследование характеристик и свойств программы с помощью различных инструментов разработчика.

Профилирование, построение покрытия кода, анализ корректности программ, анализ свойств программ (зависимости по данным, профиль обращений к памяти).

Оптимизационные отчеты компиляторов, Gprof, Gcov, Intell Parallel Studio, SAPFOR.

Снижение сложности программирования

Исследование характеристик и свойств программы с помощью различных инструментов разработчика.

Пример. *Применение SAPFOR и динамического отладчика DVM-системы при ручном распараллеливании задачи моделирования распространения упругих волн в средах со сложной 3D геометрией.*

- ~ 3000 строк кода в фиксированном формате на Fortran 95
- > 100 динамических массивов, > 90 циклов

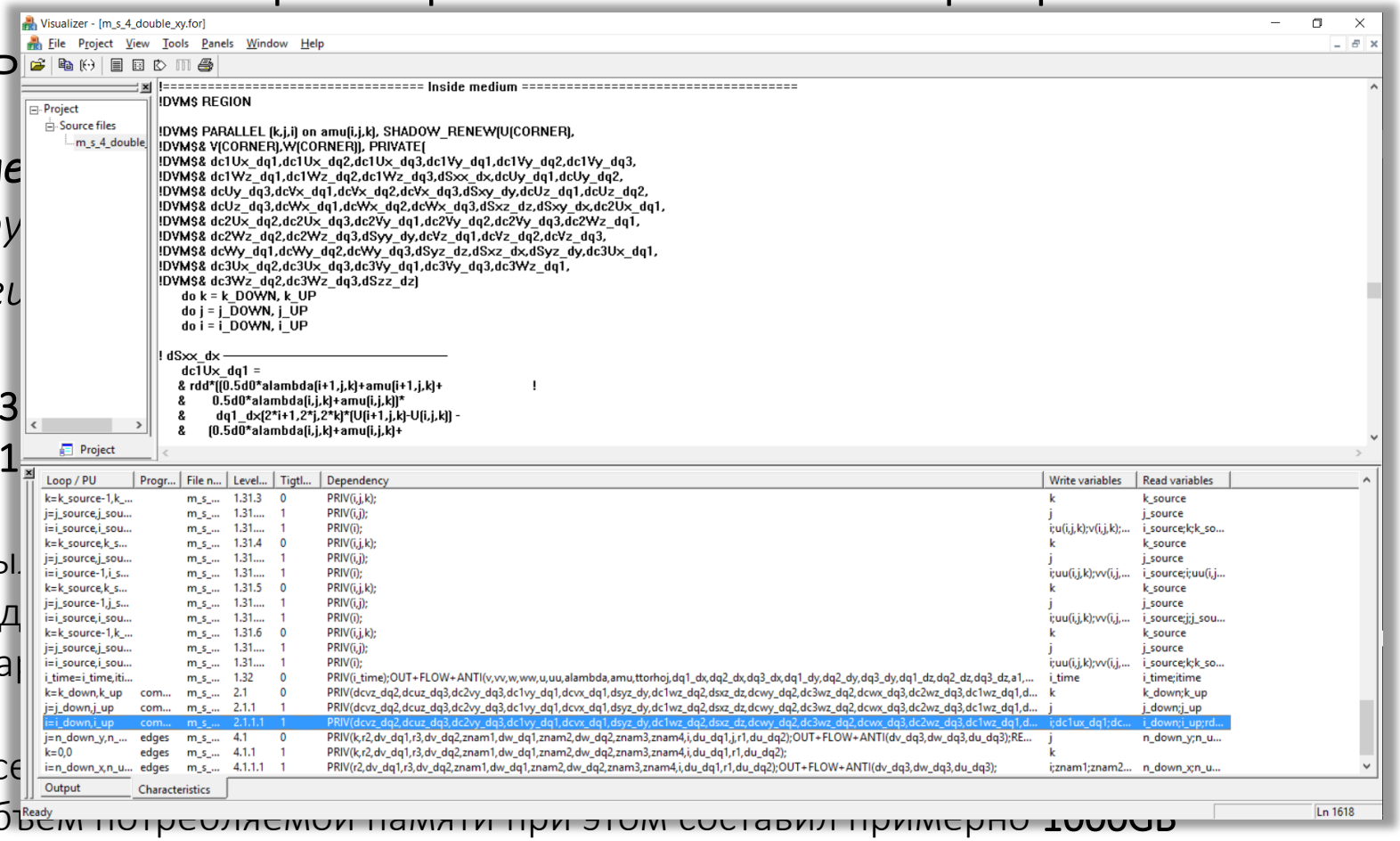
- Было добавлено 79 DVM-директив
- 1 день для распараллеливания и 2 дня на получение результатов
- Параллельное выполнение на гибридном вычислительном кластере с GPU

- Всего было задействовано 480 ядер CPU и 80 GPU (40 узлов кластера К100), объем потребляемой памяти при этом составил примерно 1000GB
- Получено практически линейное ускорение задачи:

слабая и сильная масштабируемости

Снижение сложности программирования

Исследование характеристик и свойств программы с помощь



Приме

при ру

упруги

- ~ 3
- > 1
- Бы
- 1 д
- Па
- Все

объем потребляемой памяти при этом составил примерно **1000GB**

- Получено практически линейное ускорение задачи:

слабая и сильная масштабируемости

Снижение сложности программирования

Следование определенным правилам при разработке программ на традиционных языках программирования.

Данный подход также предполагает описание свойств исходной последовательной программы, чтобы, руководствуясь ими, компилятор смог выполнить необходимые преобразования и определить соответствующие спецификации параллелизма самостоятельно.

Пример. *Рассмотренная задача моделирования распространения упругих волн в средах со сложной 3D геометрией разрабатывалась автором с применением принципов ко-дизайна и может быть распараллелена последней версией SAPFOR после задания пользователем некоторых параметров задачи.*

Снижение сложности программирования

Автоматизация анализа, преобразования и распараллеливания программ (SAPFOR).

Преобразования программ в большинстве случаев необходимо, чтобы задействовать скрытый в программах параллелизм (полная или частичная автоматизация):

- Устранение зависимостей в программе.
- Оптимизация доступа к памяти (например, при распараллеливании на GPU).
- Изменение структуры хранения данных, для обеспечения возможности их распределения.
- Изменение структуры циклов, для корректного отображения вычислений на процессы в соответствии с распределением данных, в том числе для оптимизации коммуникаций и избежание перераспределения данных.

В крайнем случае, от программиста может потребоваться выбор другого алгоритма, решающего ту же задачу, но обладающего большим потенциалом параллелизма.

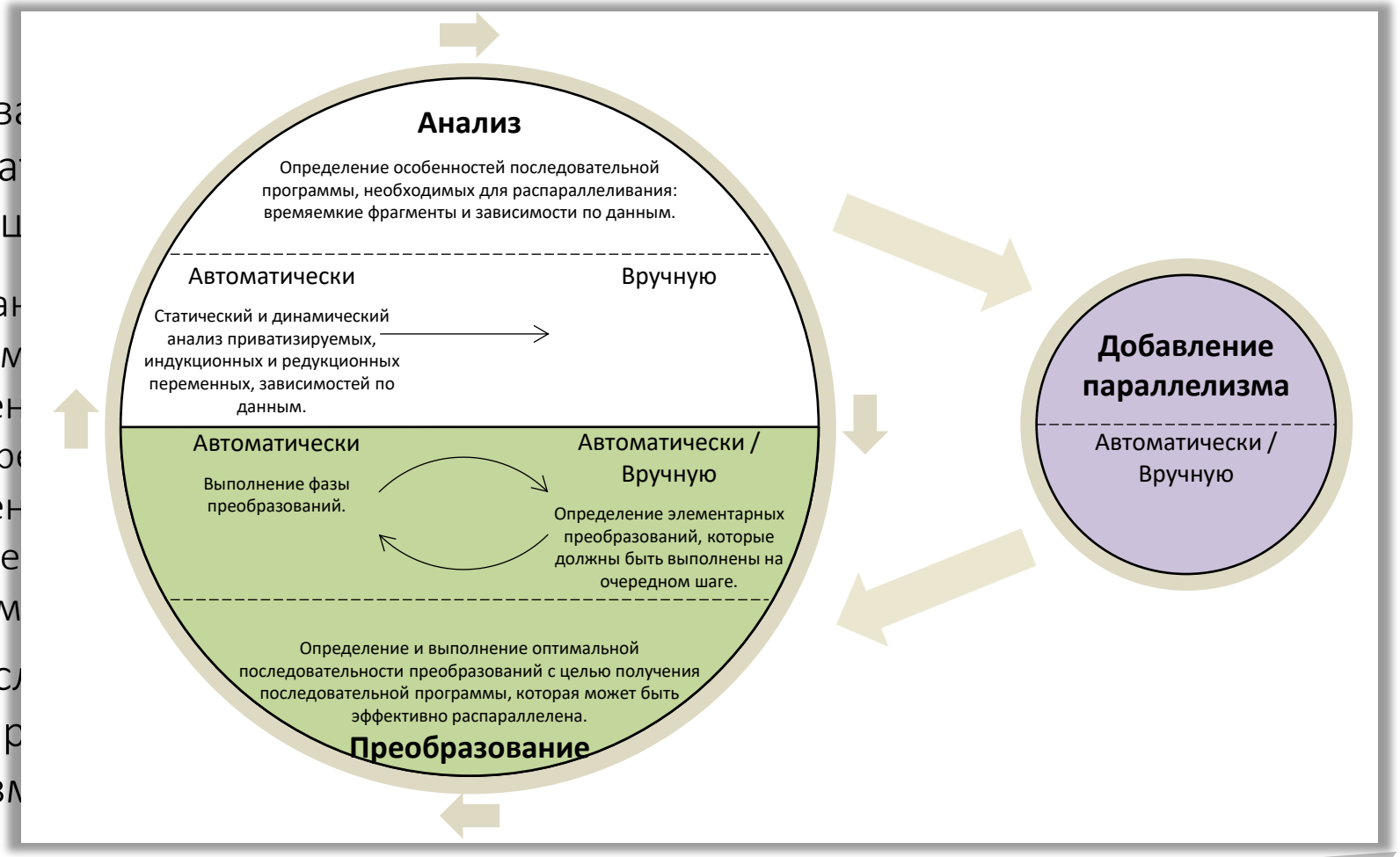
Снижение сложности программирования

Автоматизация анализа, преобразования и распараллеливания программ (SAPFOR).

Преобразова
задействоват
автоматизац

- Устран
- Оптим
- Измен
  распре
- Измен
  проце
  оптим

В крайнем сл
алгоритма, р
параллелизм



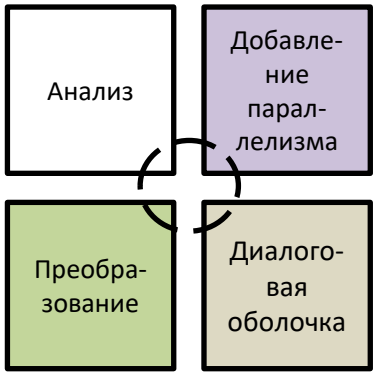
Представление процесса распараллеливания

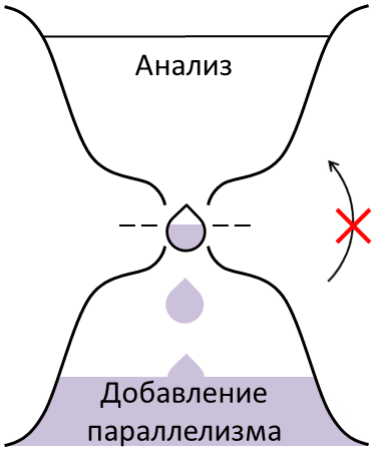
Автоматически распараллеливающий компилятор PARFOR является частью системы SAPFOR (но может выступать в роли отдельного инструмента) и отвечает за анализ программы, автоматическое принятие решений о необходимости ее преобразования, распределение данных и вычислений, оптимизацию коммуникаций.

IR PARFOR лежит в основе перечисленных подходов к снижению сложности распараллеливания программ (анализ, выявление неявного параллелизма, преобразование).

IR PARFOR обеспечивает возможность предоставления пользователю необходимой для распараллеливания информации в терминах исходной программы на протяжении всего пути распараллеливания прикладной программы пользователя.

История развития IR PARFOR





IR PARFOR = БД САПФОР

Задача – хранение информации о программе, необходимой ядру системы (эксперту) для принятия решения о распараллеливании программы, основанном на вставке директив DVM.

Заполняется анализаторами системы по исходной программе.

●●●

Попытка реализации алгоритмов анализа независящих от языка исходной программы над информацией, хранимой в БД САПФОР.

Отсутствие полной информации об исполняемых операторах программы. Для операторов сохранялась семантика доступа к памяти программы, но семантика исполняемых операций отсутствовала.

●●●

Попытка использования представления GIMPLE GCC совместно с существующей БД САПФОР была первым шагом к использованию единого внутреннего представления.

Несмотря на избирательность хранимой в БД САПФОР информации, для ее корректного заполнения требовалось практически полное восстановление исходного кода программы, что в общем случае невозможно.

●●●

Исходная программа не является частью БД САПФОР, что приводит к невозможности автоматизации преобразований в рамках единого процесса распараллеливания.

Представление процесса распараллеливания

Необходимость расширения возможностей PARFOR потребовало изменения подходов к организации IR PARFOR.

Процесс распараллеливания программы – поиск оптимизационной последовательности проходов, обеспечивающей переход от последовательной версии программы к эффективной параллельной версии.

- Проходы анализа (analysis) собирают информацию о программе.
- Проходы преобразования (transformation) используют собранную информацию для «улучшения» программы.

IR PARFOR включает в себя оптимизационную последовательность проходов и совокупность данных после выполнения каждого прохода (свойства и представление, возможно измененное, программы).

IR PARFOR должно обеспечивать согласованное представление программы на разных уровнях абстракции:

- высокоуровневое представление в соответствии с исходным языком для пользователя;
- низкоуровневое представление для выполнения анализа программы независимо от исходного языка.

Анализ программы
– *статический и динамический*

Исследование структуры программы (построение графа вызовов, графа потока управления, дерева циклов, дерева доминаторов и др.).

Исследование структуры памяти программы (анализ наложений по памяти (aliasing), определение размеров используемых массивов).

Определение зависимостей по данным:

- прямые, обратные, по выходу;
- приватизируемые переменные (скаляры и массивы);
- редукционные зависимости между витками цикла;
- индуктивные переменные;
- регулярные зависимости по массивам.

Принятие решений о распределении данных и вычислений, оптимизации коммуникаций.

Исследование покрытия программы (вызываемые функции, выполняемые операторы, количество их выполнения/вызова, границы циклов).

Профилирование программы.

Преобразование программы

С целью повышения качества проводимого анализа и для исследования информационной структуры распараллеливаемой программы:

- может выполняться над внутренним представлением программы в системе;
- нет необходимости преобразовывать исходную программу соответствующим образом.

До преобразования	После преобразования

1. void copy(float *A, int N, int M) {
2. int X;
3. for (int I = 0; I < N; ++I) {
4. X = I + M;
5. A[I] = A[X];
6. }
7. }
	1. void copy(float *A, int N, int M) {
	2. for (int I = 0; I < N; ++I) {
	3. A[I] = A[I + M];
	4. }
	5. }

С целью повышения качества распараллеливания программы:

- приводит к изменению свойств программы;
- выполняется на уровне исходного языка (Си/Фортран);
- позволяет программисту оценить принимаемые системой решения и внести корректировки в программу.

The LLVM Compiler Infrastructure

LLVM (Low Level Virtual Machine) – набор модулей и инструментов для разработки компиляторов.

Проект с открытым исходным кодом, который начинался как исследовательский проект, а сейчас активно развивается и используется многими крупными компаниями (Apple, Intel, NVIDIA, PGI и др.). LLVM включает в себя много разных проектов, направленных на распараллеливание, оптимизацию, анализ программ и поддерживает различные языки программирования.

В основе лежит независящее от исходного языка программирования внутреннее представление LLVM IR (Intermediate Representation), в которое с помощью различных front-end переводятся программы, написанные на различных языках программирования (C/C++/Fortran/Ada/Go и др.).

LLVM IR не зависит от входного языка и позволяет унифицировать проводимые анализы и оптимизации.

Реализация IR PARFOR

Clang + LLVM (для C/C++) и Sage или Flang + LLVM (для Fortran)

Гарантированная поддержка в LLVM всех существующих стандартов языков C/C++/Fortran.

Понятие прохода САПФОР хорошо согласуется с архитектурой LLVM, которая позволяет реализовать отдельные виды анализа в виде проходов, а также задействовать уже существующие в LLVM проходы.

Хорошо структурированный, спроектированный и документированный исходный код LLVM.

Для выполнения преобразований на уровне исходного кода используются возможности Clang или Sage/Flang.

Согласованность представлений программы достигается за счет поддержания точек синхронизации между AST (Clang/Sage/Flang) и LLVM IR благодаря наличию отладочной информации:

- дерево циклов;
- переменные программы;
- обращения к массивам.

Нет необходимости полного соответствия AST и LLVM IR, что позволяет выполнять преобразования необходимые для анализа программы только на уровне LLVM IR.

Три уровня абстракции

```
int f(int x, int y) {
  return x + y;
}
```

```
`-FunctionDecl f 'int (int, int)'
  |-ParmVarDecl x 'int'
  |-ParmVarDecl y 'int'
  `-CompoundStmt
    `-ReturnStmt
      `-BinaryOperator 'int' '+'
        |-ImplicitCastExpr 'int' <LValueToRValue>
        | `-DeclRefExpr 'int' lvalue ParmVar 'x' 'int'
        `-ImplicitCastExpr 'int' <LValueToRValue>
          `-DeclRefExpr 'int' lvalue ParmVar 'y' 'int'
```

AST

LLVM IR

```
define i32 @f(i32 %x, i32 %y) {
entry:
  call void @llvm.dbg.value(metadata i32 %y, i64 0, metadata !11, metadata !12)
  call void @llvm.dbg.value(metadata i32 %x, i64 0, metadata !14, metadata !12)
  %add = add nsw i32 %x, %y
  ret i32 %call
}
```

```
...
!7 = distinct !DISubprogram(name: "f", scope: !1, file: !1, l
...
!10 = !DIBasicType(name: "int", size: 32, encoding: DW_ATE_
!11 = !DILocalVariable(name: "y", arg: 2, scope: !7, file: !1, line: 3, type: !10)
...
!13 = !DILocation(line: 3, column: 18, scope: !7)
!14 = !DILocalVariable(name: "x", arg: 1, scope: !7, file: !1, line: 3, type: !10)
```

DWARF

Три уровня абстракции

```
int f(int x, int y) {
  return x + y;
}
```

```
`-FunctionDecl f 'int (int, int)'
  |-ParmVarDecl x 'int'
  |-ParmVarDecl y 'int'
  `-CompoundStmt
    `-ReturnStmt
      `-BinaryOperator 'int' '+'
        |-ImplicitCastExpr 'int' <LValueToRValue>
        | `-DeclRefExpr 'int' lvalue ParmVar 'x' 'int'
        `-ImplicitCastExpr 'int' <LValueToRValue>
          `-DeclRefExpr 'int' lvalue ParmVar 'y' 'int'
```

AST

LLVM IR

```
define i32 @f(i32 %x, i32 %y) {
entry:
  call void @llvm.dbg.value(metadata i32 %y, i64 0, metadata !11, metadata !12)
  call void @llvm.dbg.value(metadata i32 %x, i64 0, metadata !14, metadata !12)
  %add = add nsw i32 %x, %y
  ret i32 %call
}
```

```
...
!7 = distinct !DISubprogram(name: "f", scope: !1, file: !1, l
...
!10 = !DIBasicType(name: "int", size: 32, encoding: DW_ATE_
!11 = !DILocalVariable(name: "y", arg: 2, scope: !7, file: !1, line: 3, type: !10)
...
!13 = !DILocation(line: 3, column: 18, scope: !7)
!14 = !DILocalVariable(name: "x", arg: 1, scope: !7, file: !1, line: 3, type: !10)
```

DWARF

Дерево псевдонимов

Отражает структуру обращений к памяти в исходной программе.

- Обобщает информацию об участках памяти уровня LLVM IR до переменных программы на языке высокого уровня.
- Отражает иерархическую структуру типов языка высокого уровня (элемент структуры -> вся структура, элемент массива –> весь массив, int -> long).
- Не зависит напрямую от используемых front-end, так как использует отладочную информацию в формате DWARF вместо абстрактного синтаксического дерева.
- Может быть уточнено за счет преобразований уровня LLVM IR и последующего выполнения результатов анализа.

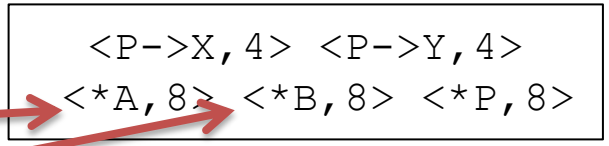
Дерево псевдонимов

```
struct S {float X; float Y;};
void foo(struct S * restrict A,
struct S * restrict B) {
  struct S *P;
  P = A;
  P->X = 0;
  P->Y = 0;
  P = B;
  P->X = 0;
  P->Y = 0;
}
```

```
<P->X,4> <P->Y,4>
<*A,8> <*B,8> <*P,8>
```

Дерево псевдонимов

```
struct S {float X; float Y;};
void foo(struct S * restrict A,
struct S * restrict B) {
  struct S *P;
  P = A;
  P->X = 0;
  P->Y = 0;
  P = B;
  P->X = 0;
  P->Y = 0;
}
```



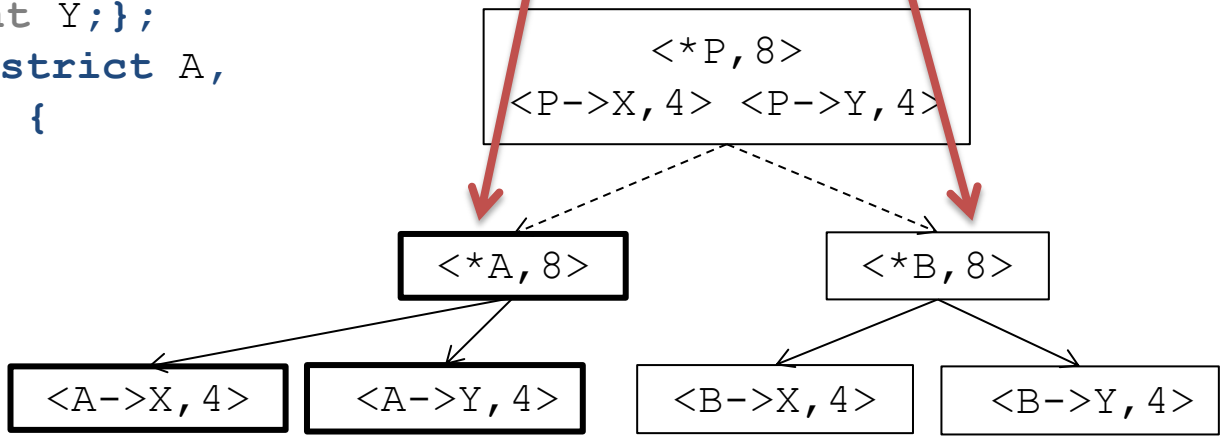
•••

Преобразование на уровне LLVM IR

```
struct S {float X; float Y;};
void foo(struct S * restrict A,
struct S * restrict B) {

  A->X = 0;
  A->Y = 0;

  B->X = 0;
  B->Y = 0;
}
```



Заключение

IR PARFOR использует:

- LLVM в качестве инструментальной базы для анализа программ;
- абстрактное синтаксическое дерево для программы на соответствующем языке (Fortran (Flang/Sage) или C (Clang)) для преобразования программ на уровне исходного кода.

IR PARFOR позволяет:

- распространить результаты анализа, полученные для программы в виде LLVM IR, на программу на исходном языке высокого уровня;
- повысить качество анализа исходной программы за счет выполнения преобразований LLVM IR, не затрагивающих структуру памяти исходной программы.

Спасибо за внимание

НАУЧНЫЙ СЕРВИС В СЕТИ ИНТЕРНЕТ 2018

KIAM RAS DVM SYSTEM

http://dvm-system.org

ОРДЕНА ЛЕНИНА им.М.В.КЕЛДЫША РАН