МЕЖДУНАРОДНАЯ НАУЧНАЯ КОНФЕРЕНЦИЯ

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ ТЕХНОЛОГИИ 2017

Преобразование программ на высокоуровневом языке программирования на основе результатов анализа низкоуровневого представления программ в системе SAPFOR

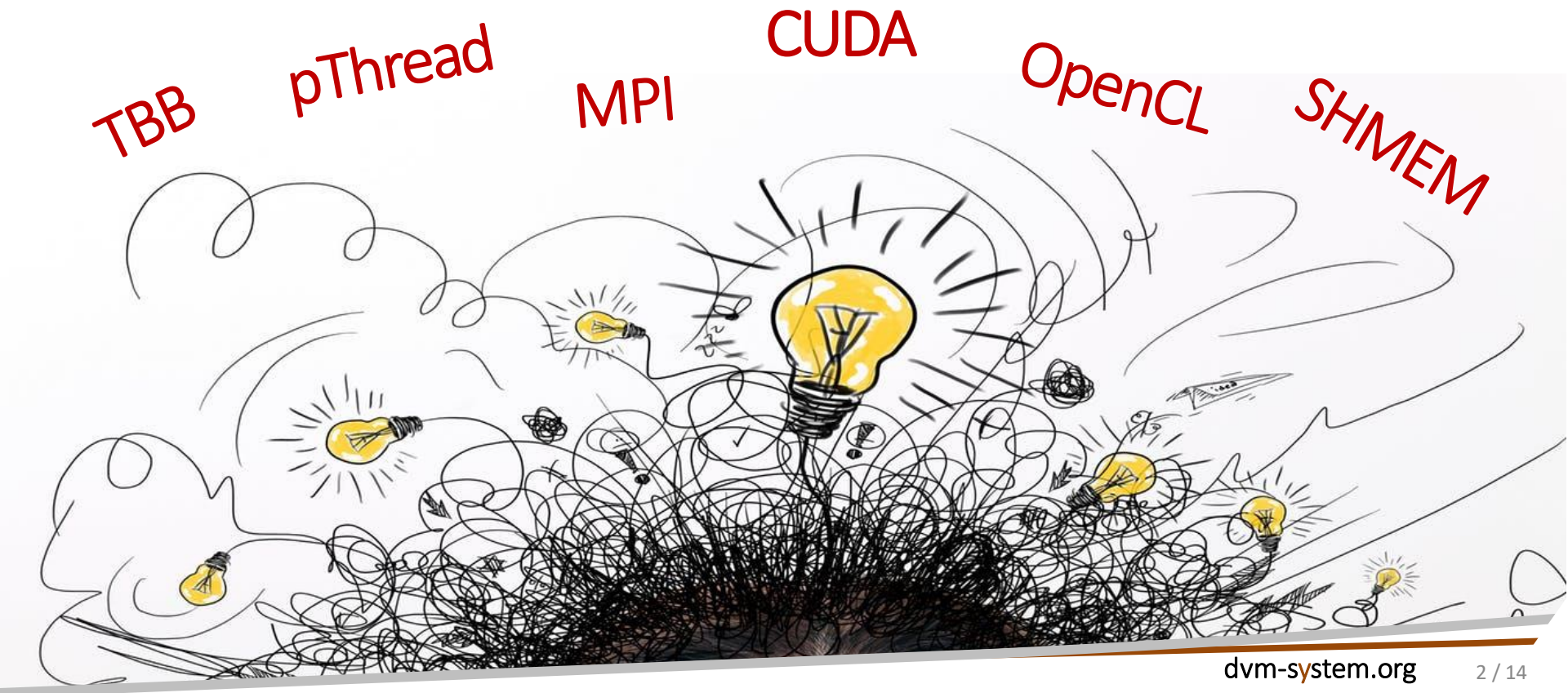
4 апреля 2019 г. | Калининград

Катаев Н.А., Козырев В.И.

Институт прикладной математики им. М.В. Келдыша РАН,
Факультет вычислительной математики и кибернетики МГУ
им. М.В. Ломоносова

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.



Средства параллельного программирования

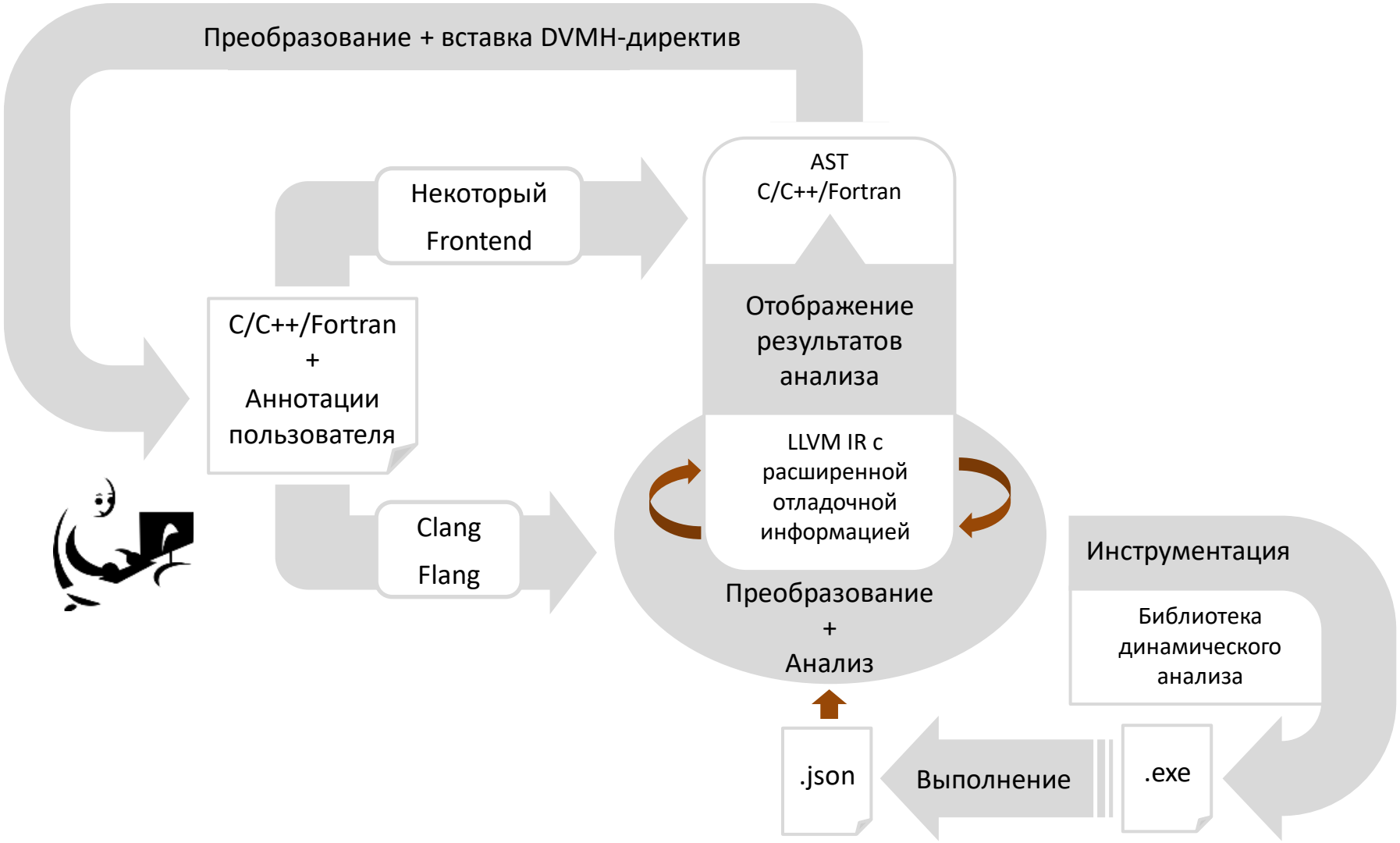
SAPFOR (System FOR Automated Parallelization)

- Исследование характеристик и свойств программы (профилирование, анализ зависимостей по данным и др.).
- Автоматическое распараллеливание «хорошо» написанных потенциально параллельных программ.

  Следование определенным правилам при разработке программ на традиционных языках программирования, дополнительное описание свойств программ.
- Автоматизированное преобразование программ к потенциальном параллельному виду.

  Устранение зависимостей в программе, оптимизация доступа к памяти, изменение структуры хранения данных и структуры вычислений (подстановка процедур и переменных, преобразование циклов и др.).

Внутреннее устройство SAPFOR



Цели выполнения преобразований

Повышение 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;
- невидимо для пользователя.

До	После
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. `}`

Данное преобразование может быть скрыто от пользователя.

Повышение 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.

Source-to-source преобразование программы

1. Исследование структуры преобразуемого участка программы с помощью Clang AST.
2. Формирование графа потока данных на основе LLVM IR построенного непосредственно по Clang AST, построение структуры используемой памяти (alias analysis).
3. Преобразование LLVM IR для повышения качества проводимого анализа (замена переменных агрегатных типов скалярными переменными, размещение скалярных переменных на регистрах, подстановка скалярных переменных, упрощение структуры циклов и графа потока управления, объединение инструкций и др.).
4. Уточнение структуры используемой памяти, определение и классификация зависимостей по данным, исследование графа потока данных, построенного на шаге (2).
5. Отображение полученных для преобразованного LLVM IR результатов на структуры Clang AST, описывающие исходную программу.
6. Выполнение преобразования исходной программы с помощью Clang AST.

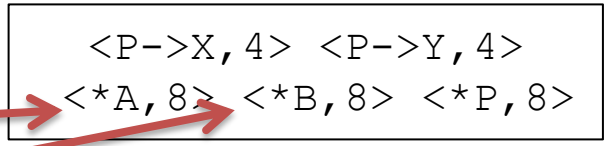
Дерево псевдонимов

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: пример

```
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;
}
```



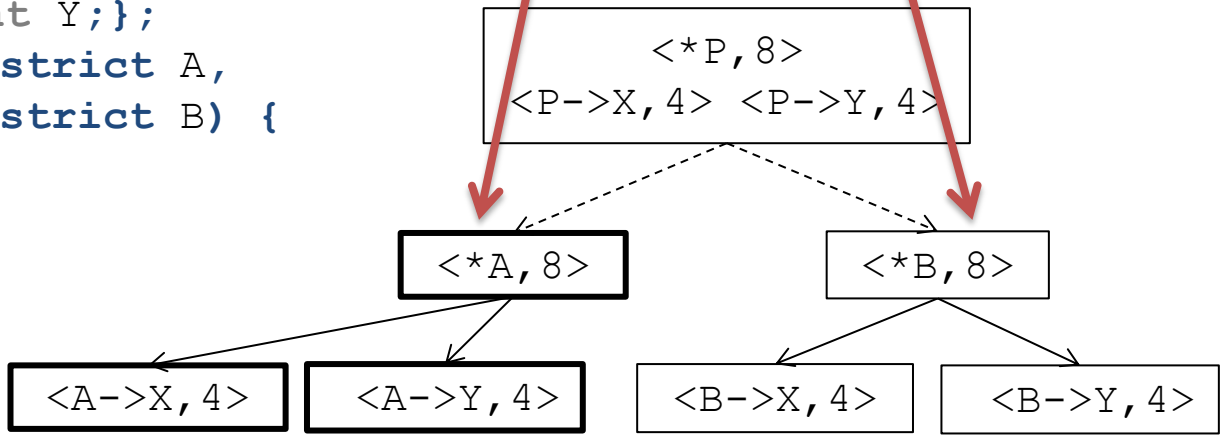
•••

Преобразование 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;
}
```



Пример: внос инварианта в тело цикла

Позволяет выполнить объединение двух циклов (loop fusion), которые не являются соседними из-за того, что между ними присутствует некоторый код.

```
for(...) { body1 }
code;
for(...) { body2 }
```

→

```
for(...) { body1 }
for(...) { code; body2 }
```

→

```
for(...) {
  body1 code; body2
}
```

В свою очередь объединение циклов снижает накладные расходы на синхронизацию и способствует повышению локальности использования данных.

Сспособствует приведению гнезда циклов к тесно вложенному виду.

```
for(...) {
  code;
  for(...) { body2 }
}
```

→

```
for(...) {
  for(...) {
    code;
    body2
  }
}
```

Тесная вложенность позволяет повысить уровень параллелизма в программе, например, организовав конвейерное выполнение циклов, содержащих прямую зависимость по данным с фиксированным расстоянием.

Пример: внос инварианта в тело цикла

Определение не тесно вложенных циклов программы.

Clang AST

Определение циклов представленных в каноническом виде.

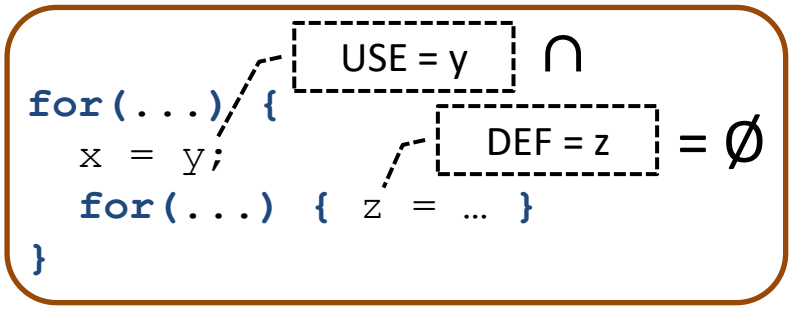
Clang AST + LLVM IR

- `for (init-expr; test-expr; incr-expr) structured-block`
- Границы и шаг цикла представлены выражениями, инвариантными для цикла.
- Переменная-счетчик цикла однократно увеличивается на каждой итерации в заголовке цикла.

Проверка допустимости внесения инструкций в тело цикла.

LLVM IR

- Основана на анализе графа потока данных.
- Использует alias analysis.



Выполнение преобразования **исходной программы.**

Clang AST

Преобразование Fortran-программ

Flang может быть использован для генерации LLVM IR и анализа программ, *но* не подходит для source-to-source преобразований исходного кода.

SAPFOR позволяет использовать разные фронтенды для анализа и преобразования программ.

Стыковка выполняется за счет использования отладочной информации, представленной в LLVM в формате DWARF.

Для преобразования Фортран-программ используется фронтенд, входящий в состав компилятора с языка Fortran-DVMH.

Lawrence Livermore National Laboratory

Home / News / NNSA, national labs team with Nvidia to develop open-source Fortran compiler technology

Nov. 13, 2015

The Department of Energy's National Nuclear Security Administration and its three national labs will work with NVIDIA's PGI® software to create an open-source Fortran compiler designed for integration with the widely used LLVM compiler infrastructure. (Download Image)

NNSA, national labs team with Nvidia to develop open-source Fortran compiler technology

Заключение

В SAPFOR программа представлена на двух уровнях, соответствие между которыми устанавливается за счет использования отладочной информации:

- LLVM IR для исследования свойств программы;
- ASTдля выполнения преобразований на уровне исходного кода.

Допускается использование разных фронтендов для генерации LLVM IR с целью анализа и генерации AST с целью преобразования.

LLVM IR обеспечивает возможность использования единых алгоритмов анализа для программ на разных языках (C/C++ - Clang, Fortran – Flang).

Проверки допустимости выполнения source-to-source преобразований, связанные с особенностями обращений к памяти (зависимости по данным, анализы потока данных), могут быть выполнены над LLVM IR.

Дерево псевдонимов позволяет использовать невидимые для пользователя преобразования LLVM IR, чтобы повысить точность описания используемой памяти.

Спасибо за внимание

http://dvm-system.org

dvm@keldysh.ru