ВСЕРОССИЙСКАЯ НАУЧНАЯ КОНФЕРЕНЦИЯ
НАУЧНЫЙ СЕВИС В СЕТИ ИНТЕРНЕТ

Восстановление обращений к многомерным массивам в системе SAPFOR

25 сентября 2019 г. | Абрау-Дюрсо

Н.А. Катаев[1], В.Н. Василькин[2]

[1]Институт прикладной математики им. М.В. Келдыша РАН

[2]Московский государственный университет им. М.В. Ломоносова

Мотивация

```
float A[N][N];
for (int I = 1; I < N - 1; I++)
  for (int J = 1; J < N - 1; J++)
    A[I][J] = A[I][J - 1];
```

Анализ зависимостей по данным сводится к поиску целочисленного решению системы линейных неравенств, нахождение такого решения является NP-трудной задачей.

$$\begin{cases} I_1 * N + J_1 = I_2 * N + J_2 - 1 \\ 0 < I_{1,2} < N - 1 \\ 0 < J_{1,2} < N - 1 \end{cases}$$

Если N не является константой времени компиляции, то задача перестает быть линейной.

Знание многомерной структуры массива позволяет во многих случаях снизить сложность решаемой задачи.

$$\begin{cases} I_1 = I_2 \\ J_1 = J_2 - 1 \\ 0 < I_{1,2} < N - 1 \\ 0 < J_{1,2} < N - 1 \end{cases}$$

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 потенциальному 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.).

SAPFOR Intermediate Representation (IR)

Требования к SAPFOR IR :

- Поддержка языков программирования Фортран (95 и выше) и Си (99 и выше).
- Возможность реализовать различные алгоритмы анализа (определение зависимостей по данным, определение индуктивных и редукционных переменных, определение возможных пересечений по памяти (alias analysis) и др.), которые необходимы для распараллеливания программ.
- Поддержка преобразований исходного кода (source-to-source).
- Возможность исследования больших вычислительных комплексов. В том числе, поддержка стандартных опций компилятора, возможность интеграции с инструментами автоматизации сборки.

При выборе используемых инструментов были исследованы следующие инфраструктуры компиляторов: **Sage, Cetus, Rose, Ops, GCC, LLVM.**

The LLVM Compiler Infrastructure

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

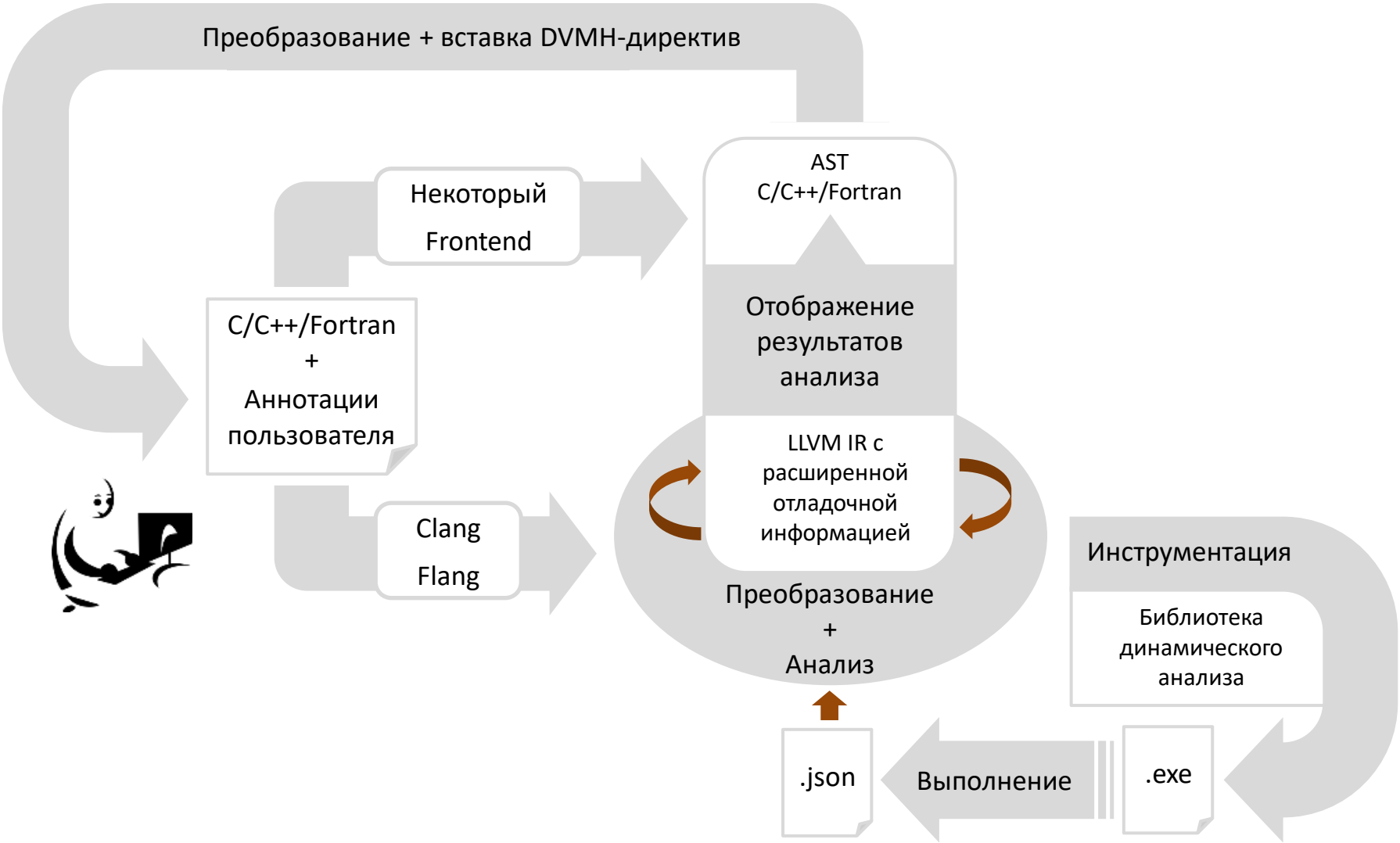
Проект с открытым исходным кодом, который начинался как исследовательский проект, а сейчас активно развивается и используется многими крупными компаниями (Apple, Intel, NVIDIA, PGI и др.). LLVM включает в себя много разных проектов, направленных на распараллеливание, оптимизацию, анализ программ и поддерживает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

http://llvm.org/

Внутреннее устройство SAPFOR



Постановка задачи

Восстановление формы массивов, которые были многомерными в исходном коде, но оказались линеаризованы при построении LLVM IR.

Делинеаризованное представление массивов должно совпадать с представлением многомерных массивов в исходном коде программы.

С целью повышения качества проводимого в SAPFOR анализа: обеспечить возможность исследования преобразованного LLVM IR, избежав необходимости преобразования исходного кода программы.

До преобразования LLVM IR	**После преобразования LLVM IR**
1. float A[N][N]; 2. for (int I = 1; I < N - 1; ++I) 3. for (int J = 1; J < N - 1; ++J) { 4. int X = J - 1; 5. A[I][J] = A[I][X]; 6. }	1. float A[N][N]; 2. for (int I = 1; I < N - 1; ++I) 3. for (int I = 1; I < N - 1; ++I) { 4. A[I][J] = A[I][J - 1]; 5. } 6. }
$\begin{cases} I_1 = I_2 \\ J_1 = X \\ 0 < I_1 < N \end{cases}$ *Проблема: система зависит от переменной не связанной с циклом.*	$\begin{cases} I_1 = I_2 \\ J_1 = J_2 - 1 \\ 0 < I_{1,2} < N \end{cases}$

Идея алгоритма

В линеаризованной форме обращение $A[S_0]\dots[S_{N-1}]$ к массиву $A[D_0]\dots[D_{N-1}]$ имеет вид:

$$A + S_0 * D_1 * \cdots D_{N-I} + \cdots + S_{N-1}$$

D_I – размер измерения массива с номером I,
S_I – индексное выражение с номером I,
$0 \le I \le N-1$

Чтобы вычислить размер измерения массива с номером I необходимо найти наибольший общий делитель слагаемых, расположенных слева от слагаемого I, и разделить его на произведение вычисленных ранее размеров измерений от I+1 до N-1.

$$C_{N-1} * D_{N-1} = GCD(S_0 * D_1 * \cdots * D_{N-1}, \dots, S_{N-2} * D_{N-1})$$
$$C_I * D_I = GCD(S_0 * D_1 * \cdots * S_{I-1} * D_I * \cdots * D_{N-1}) / (D_{I+1} * \cdots * D_{N-1})$$
$$0 \le I \le N-1$$

Необходимо определить:

- правильное количество слагаемых (равное количеству измерений в массиве);
- упорядочить их в соответствии с порядком измерений;
- определить значение коэффициента C_I.

Реализация доступная в LLVM (Polly)

Основная задача Polly – оптимизация циклов и повышение локальности используемых данных, а также распараллеливание для систем с общей памятью (OpenMP, GPU).

Оптимизация выполняется над LLVM IR поэтому не требуется соотнесение делинеаризованных массивов с объектами исходного кода.

- Термами считаются только те слагаемые, которые содержат произведение индексной переменной объемлющего цикла на переменные программы.
- Константные множители игнорируются, коэффициента C_I считается равным 1.
- Упорядочивание термов выполняется в соответствии с количеством входящих в них множителей.

Оптимизации подвергаются отдельные циклы, и не требуется согласованная делинеаризация массивов в рамках всей программы.

Делинеаризованное представление массива может не соответствовать представлению массива в исходной программе, особенно если часть измерений массивов имеет фиксированные размеры.

Обращения к массивам в LLVM IR

Инструкция **getelementptr** вычисляет адреса элемента агрегатного типа:

<result> = getelementptr <ty>, <ty>* <ptrval>{, <ty> <idx>}*

Правила по которым выполняется индексация зависят от типа индексируемого объекта.

Массивы с фиксированным размером измерений представлены в LLVM IR явным образом, например, тип **[100 x [200 x float]]** может быть использован для объявления в LLVM IR массива содержащего **100*200** элементов.

```
; A[%idxprom][%idxprom1]
getelementptr [100 x [200 x float]],
 [100 x [200 x float]]* @A, i64 0, i64 %idxprom, i64 %idxprom1
```

Если размер измерения вычисляется во время выполнения программы, то параметрами инструкции **getelementptr** являются смещения относительно начала участка памяти:

```
; float A[M][N][2]; A[I][J][1] = ...
; A + (I * N) * 2
%6 = zext i32 %I.0 to i64
%7 = mul nuw nsw i64 %6, %1
%arrayidx11 = getelementptr [2 x double], [2 x double]* %vla, i64 %7
%8 = zext i32 %J.0 to i64
; A + (I * N) * 2 + J
%arrayidx13 = getelementptr [2 x double], [2 x double]* %arrayidx11, i64 %8
; A + (I * N) * 2 + J + 1
%arrayidx14 = getelementptr [2 x double], [2 x double]* %arrayidx13, i64 0, i64 1
```

Реализация в SAPFOR

Количество термов в обращении (размерность массива) определяется на основе количества параметров инструкций getelemntptr: вычисляется наибольшая размерность среди всех обращений к массиву внутри анализируемой функции.

> *Для вычисления количества термов не требуется связь обращения к массиву с циклом => могут обрабатываться обращения вне циклов.*

При наличии используется отладочная информация, содержащая количество измерений массива и их размеры.

↓

Определение размеров измерений:

$$\boldsymbol{D_I = GCD(S_0 * D_1 * \cdots * S_{I-1} * D_I * \cdots * D_{N-1}) \,/\, (D_{I+1} * \cdots * D_{N-1})}$$

Наибольший общий делитель вычисляется среди термов, полученных для всех обращений к элементам заданного массива внутри анализируемой функции.

↓

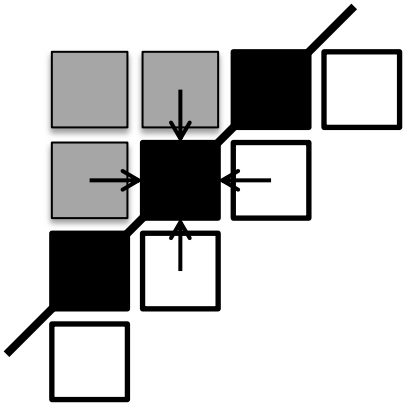
Вычисляется индексное выражение, соответствующее каждому терму: терм делится на произведение размеров измерений массива, участвующих в его формировании.

Применение в SAPFOR

```
float Eps = 0.f;

for (int I = 1; I < N - 1; I++)
  for (int J = 1; J < N - 1; J++)
    float S;
    S = A[I][J];
    A[I][J] = (W / 4) * (A[I - 1][J] + A[I + 1][J] +
      A[I][J - 1] + A[I][J + 1]) + (1 - W) * A[I][J];
    Eps = Max(fabs(S - A[I][J]), Eps);
}
```

Конвейерное выполнение



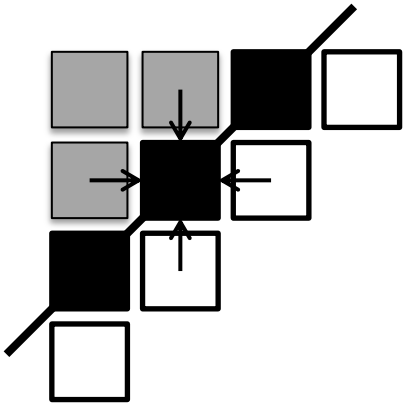
tsar -print-only=da-di -print-step=3 sor.c -finbounds-subscripts

```
private:
 <S:9:15, 4>
anti:
 <*A:5:36, ?>:[1,1]
flow:
 <*A:5:36, ?>:[1,1]
induction:
 <J:8:14, 4>:[Int,1,,1]
reduction:
 <Eps:6:9, 4>:max
read only:
     <A:5:36, 8> | <I:7:12, 4> | <N:5:25, 4> | <W:5:18, 4>
```

Регулярная зависимость по данным может быть устранена за счет конвейерного выполнения цикла: директивы **across** DVM или **ordered** OpenMP.

Применение в SAPFOR

```
float Eps = 0.f;
#pragma dvm parallel([I][J] on A[I][J]) across(A[1:1][1:1] reduction(max(Eps))
for (int I = 1; I < N - 1; I++)
  for (int J = 1; J < N - 1; J++)
    float S;
    S = A[I][J];
    A[I][J] = (W / 4) * (A[I - 1][J] + A[I + 1][J] +
      A[I][J - 1] + A[I][J + 1]) + (1 - W) * A[I][J];
    Eps = Max(fabs(S - A[I][J]), Eps);
}
```



tsar -print-only=da-di -print-step=3 sor.c -finbounds-subscripts

```
private:
 <S:9:15, 4>
anti:
 <*A:5:36, ?>:[1,1]
flow:
 <*A:5:36, ?>:[1,1]
induction:
 <J:8:14, 4>:[Int,1,,1]
reduction:
 <Eps:6:9, 4>:max
read only:
     <A:5:36, 8> | <I:7:12, 4> | <N:5:25, 4> | <W:5:18, 4>
```

Регулярная зависимость по данным может быть устранена за счет конвейерного выполнения цикла: директивы **across** DVM или **ordered** OpenMP.

Заключение

Предложенный подход к делинеаризации опирается на отладочную информацию, доступную в LLVM, и структуру инструкций, используемых для вычисления смещений относительно начала массива.

Реализация выполнена в рамках системы SAPFOR и проверена на большом количестве тестовых данных, а также тестах производительности NAS Parallel Benchmarks 3.3.1 и тестовом наборе Polybench/C the Polyhedral Benchmark suite 4.2.1.

Реализованный алгоритм используется в анализаторе системы SAPFOR для определения зависимостей по данным в программе, в том числе для определения реглярных зависимостей с целью их устранения за счет конвейерного выполнения циклов. Также данных алгоритм использован при автоматизированном построении распределения данных системой SAPFRO в модели DVMH.

Дальнейшие работы планируется направить на совместное использование предложенного подхода и подхода, реализованного в LLVM для преобразования исходного кода программы и делинеаризации явно линеаризованных массивов.

https://github.com/dvm-system

Работа выполнена при поддержке Российского фонда фундаментальных исследований, проекты 18-01-00851-а, 17-01-00820-а.

Спасибо за внимание

http://dvm-system.org