ВСЕРОССИЙСКАЯ НАУЧНАЯ КОНФЕРЕНЦИЯ
НАУЧНЫЙ СЕВИС В СЕТИ ИНТЕРНЕТ

Динамический анализ зависимостей по данным в системе SAPFOR

25 сентября 2019 г. | Абрау-Дюрсо

Н.А. Катаев[1], А.А. Смирнов[1], А.Д. Жуков[2]

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

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

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.).

Анализ программ

Статический анализ

Возможности ограничены из-за наличия указателей, косвенной индексации, параметров задачи…

Динамический анализ

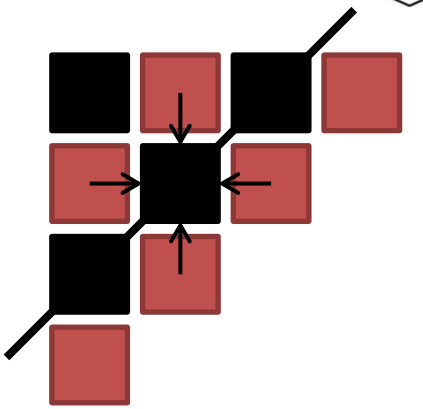
Большие накладные расходы как по времени, так и по памяти.
Невозможно перебрать все наборы входных данных.

Определение свойств программы, влияющих на распараллеливание

- зависимости по данным:
  - прямые, обратные, по выходу;
  - приватные переменные (скаляры и массивы);
  - редукционные зависимости между витками цикла;
  - индуктивные переменные;
  - регулярные зависимости по массивам;
- неявные циклы;
- время выполнения витков последовательных циклов;
- и др.

Статический анализ

```
float Eps = 0.f;
for (int I = 1; I < N - 1; I++)
  for (int J = 1; J < N - 1; J++)
    if ((I + J) % 2 == 1) {
      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);
    }
```

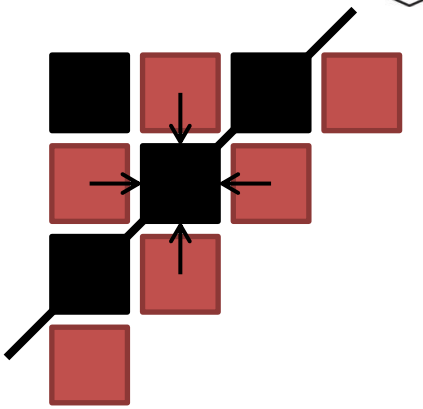


tsar -print-only=da-di -print-step=3 redblack.c

```
private:
 <S:41:15, 4>
output:
 <A, 256>
anti:
 <A, 256>
flow:
 <A, 256>
induction:
 <J:39:14, 4>:[Int,1,,1]
reduction:
 <Eps:37:9, 4>:max
read only:
 <I:38:12, 4> | <N:36:25, 4> | <W:36:18, 4>
```

Статический анализ

```
float Eps = 0.f;
for (int I = 1; I < N - 1; I++)
  for (int J = 1; J < N - 1; J++)
    if ((I + J) % 2 == 1) {
      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);
    }
```

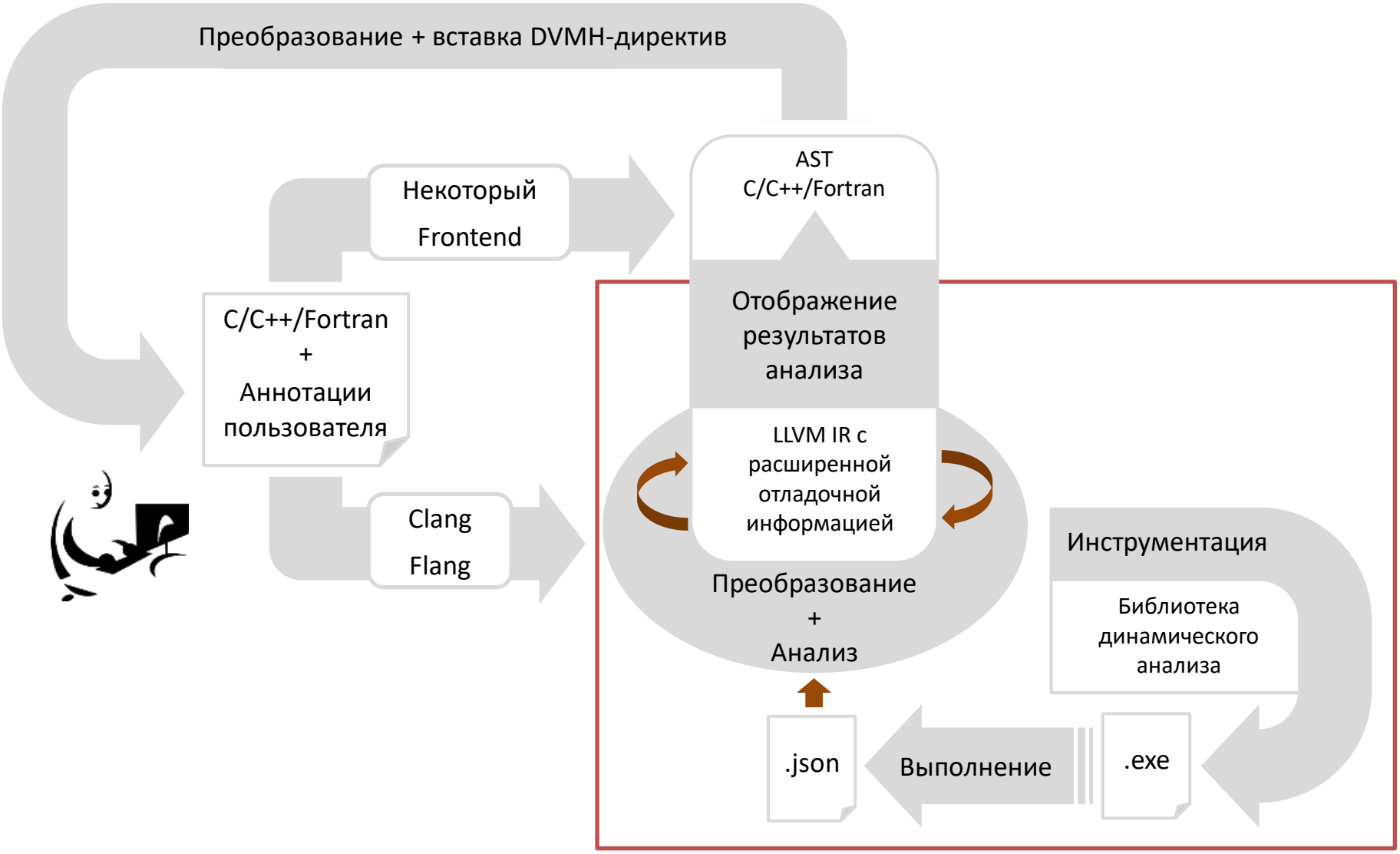


```
icc -version
icc (ICC) 19.0.2.187 20190117
Copyright (C) 1985-2019 Intel Corporation. All rights reserved.
icc -qopt-report=5 -parallel -O3 redblack.c

remark #17104: loop was not parallelized: existence of parallel dependence
remark #17106: parallel dependence: assumed FLOW dependence between A[i][j] and A[i][j-1]
remark #17106: parallel dependence: assumed ANTI dependence between A[i][j-1] and A[i][j]
remark #15344: loop was not vectorized: vector dependence prevents vectorization
remark #15346: vector dependence: assumed FLOW dependence between A[i][j] and A[i][j-1]
remark #15346: vector dependence: assumed ANTI dependence between A[i][j-1] and A[i][j]
```

```
 <J:39:14, 4>:[Int,1,,1]
reduction:
 <Eps:37:9, 4>:max
read only:
 <I:38:12, 4> | <N:36:25, 4> | <W:36:18, 4>
```

Динамический анализ в SAPFOR



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.

Архитектура анализа

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

Обрабатываемый цикл
- номер итерации (1)

Обрабатываемая функция

Стек контекстов

Хранилище результатов

Архитектура анализа

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

Обрабатываемый цикл
- номер итерации (1)

Обрабатываемый цикл
- номер итерации (1)

Обрабатываемая функция

Стек контекстов

Хранилище результатов

Архитектура анализа

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

Обрабатываемый цикл
- номер итерации (1)
- обращение по адресу A[1][0]
  - последнее чтение на итерации (1)
  - первое чтение на итерации (1)

Обрабатываемый цикл
- номер итерации (1)

Обрабатываемая функция

Стек контекстов

Хранилище результатов

Архитектура анализа

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

Обрабатываемый цикл
- номер итерации (1)
- обращение по адресу A[1][0]
- обращение по адресу A[1][1]
  - последняя запись на итерации (1)
  - первая запись на итерации (1)

Обрабатываемый цикл
- номер итерации (1)

Обрабатываемая функция

Стек контекстов

Хранилище результатов

Архитектура анализа

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

Обрабатываемый цикл
- номер итерации (2)
- обращение по адресу A[1][0]
- обращение по адресу A[1][1]

Обрабатываемый цикл
- номер итерации (1)

Обрабатываемая функция

Стек контекстов

Хранилище результатов

Архитектура анализа

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

Обрабатываемый цикл
- номер итерации (2)
- обращение по адресу A[1][0]
- обращение по адресу A[1][1]
  - последняя запись на итерации (1)
  - первая запись на итерации (1)
  - последнее чтение на итерации (2)
  - первое чтение на итерации (2)
  - прямая зависимость расстояния (1,1)

Обрабатываемый цикл
- номер итерации (1)

Обрабатываемая функция

Стек контекстов

Хранилище результатов

Архитектура анализа

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

Обрабатываемый цикл
- номер итерации (N-1)
- обращение по адресу A[1][0]
- …
- обращение по адресу A[1][N-2]

Цикл по переменной J
- прямая зависимость по переменной A расстояния (1,1)

Обрабатываемый цикл
- номер итерации (1)
- обращение по адресу A[1][0]
  - последнее чтение на итерации (1)
  - первое чтение на итерации (1)
- …

Обрабатываемая функция

Стек контекстов

Хранилище результатов

Структура собираемой информации

Представление результатов анализа в текстовом или JSON форматах.

Формат JSON файла:

```
{
 "Vars": [
    {
     "File": <path-to-file>,
     "Line": <number>,
     "Column": <number>,
     "Name": <name>,
    },
 "Loops": [
    {
     "File": <path-to-file>,
     "Line": <number>,
     "Column": <number>,
     "Private: [<var-list>],
     "UseAfterLoop": [<var-list>],
     "Flow": [{<var-id> : {"Min": <distance>, "Max": <distance>},...],
     "Anti": [{<var-id> : {"Min": <distance>, "Max": <distance>},...],
     "Output": [{<var-id> : {"Min": <distance>, "Max": <distance>},...]
    }
}
```

Список зарегистрированных переменных.
Список циклов:

- Переменные, получающие значение до использования на той же итерации цикла.
- Переменные, для которых значение полученное в цикле используется после выхода из цикла.
- Переменные, порождающие зависимость по данным (с указанием расстояния зависимости).

Использование динамического анализа

1. Получение LLVM IR для каждого файла, который должен быть проанализирован:

```
tsar <options> –emit-llvm <file1>.c  … <fileN>.c
flang <options> –S –g –emit-llvm <file1>.for … <fileN>.for
```

2. Компоновка полученных файлов, содержащих LLVM IR, с помощью инструмента llvm-link, входящего в состав LLVM:

```
llvm-link –S <file1>.ll … <fileN>.ll –o <file>.ll
```

3. Инструментация полученного единого файла (модуля LLVM IR):

```
tsar –instr-llvm <file>.ll
```

4. Компиляция инструментированного файла, компоновка его с остальными частями анализируемого проекта, а также с библиотекой динамического анализа:

```
clang <file>.ll –lda –o <file>.out
flang <file>.ll –lda –o <file>.out
```

5. Запуск инструментированной программы на выполнение (формат представления информации (Text/JSON, а также имя файла можно задать используя переменные окружения):

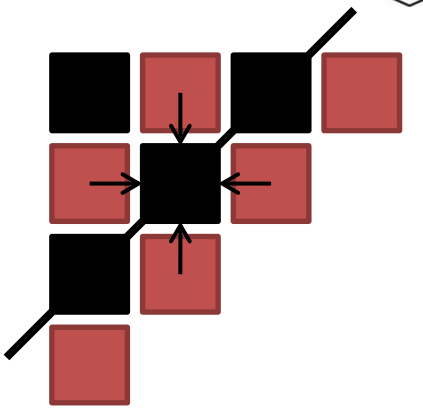
```
./a.out
```

6. Уточнение результатов статического анализа:

```
tsar <file>.ll –print-only=da-di –print-step=3 –fanalysis-use=dyna-output.json
```

Смешенный анализ

```
float Eps = 0.f;
for (int I = 1; I < N - 1; I++)
  for (int J = 1; J < N - 1; J++)
    if ((I + J) % 2 == 1) {
      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 -instr-llvm redblack.c
clang redblack.ll -lda
tsar -print-only=da-di -print-step=3 redblack.c -fanalysis-use=dyna-output.json

shared:
 <A, 256>
private:
 <S:41:15, 4>
induction:
 <J:39:14, 4>:[Int,1,,1]
reduction:
 <Eps:37:9, 4>:max
read only:
 <I:38:12, 4> | <N:36:25, 4> | <W:36:18, 4>
```

```
output:
  <A, 256>
anti:
  <A, 256>
flow:
  <A, 256>
```

Выборочная инструментация

Скалярные переменные, не участвующие в адресной арифметике в большинстве случаев могут быть проанализированы статически. Для зависимостей по данным переменным может быть определен способ их устранения, например, за счет использования редукционных операций или приватизации соответствующих данных.

Данные переменные могут быть проигнорированы в процессе динамического анализа и оптимизированы средствами LLVM.

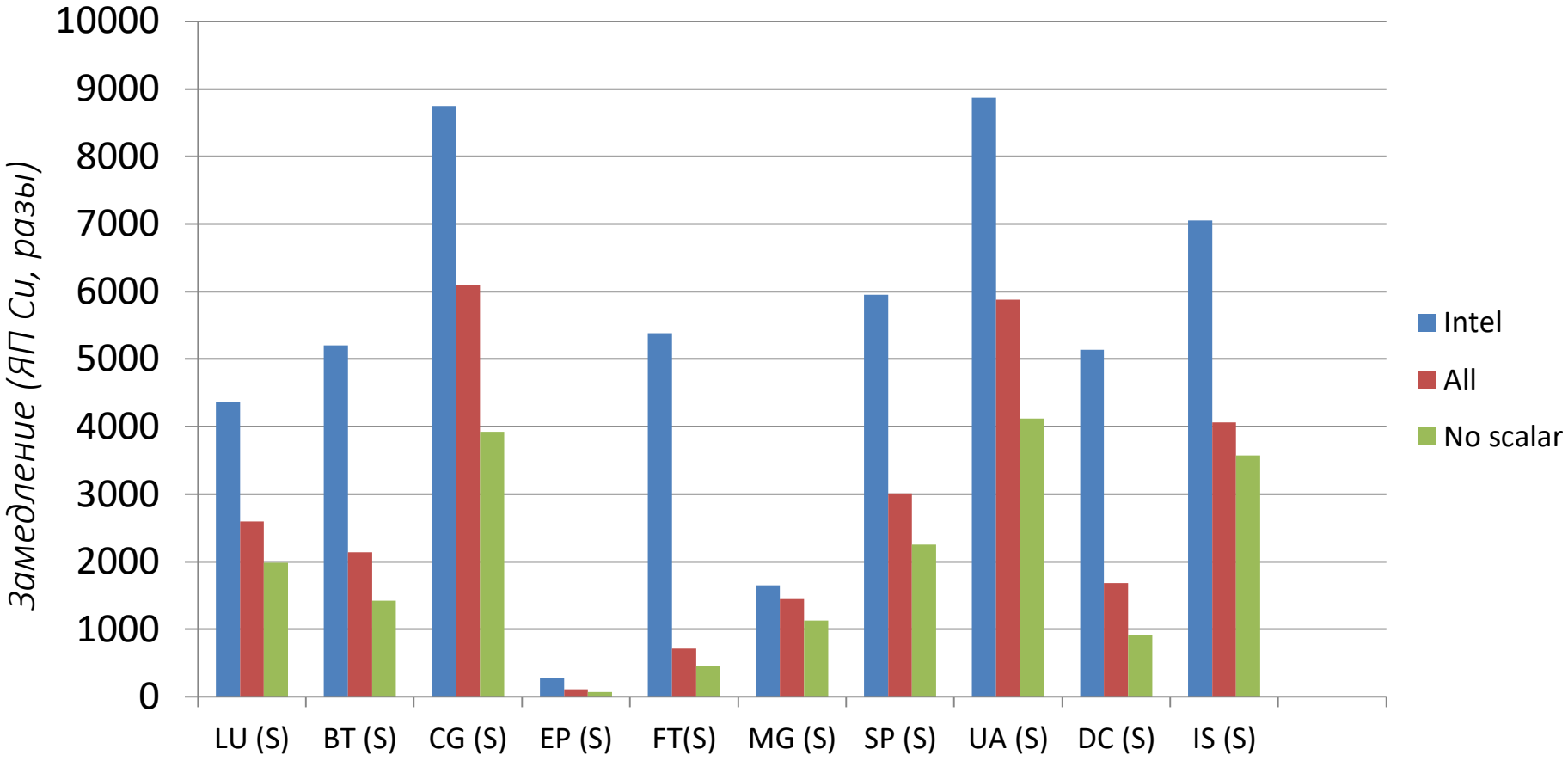
```
opt <file>.ll –sroa –o <file>.ll
tsar –instr-llvm <file>.ll
```

Могут быть выбраны функции, начиная с которых должен выполняться динамический анализ (инструментация). В данном случае будут проанализированы указанные функции, а также все функции, которые из них вызываются.

```
tsar –instr-llvm <file>.ll  -instr-start=<function-list>
```

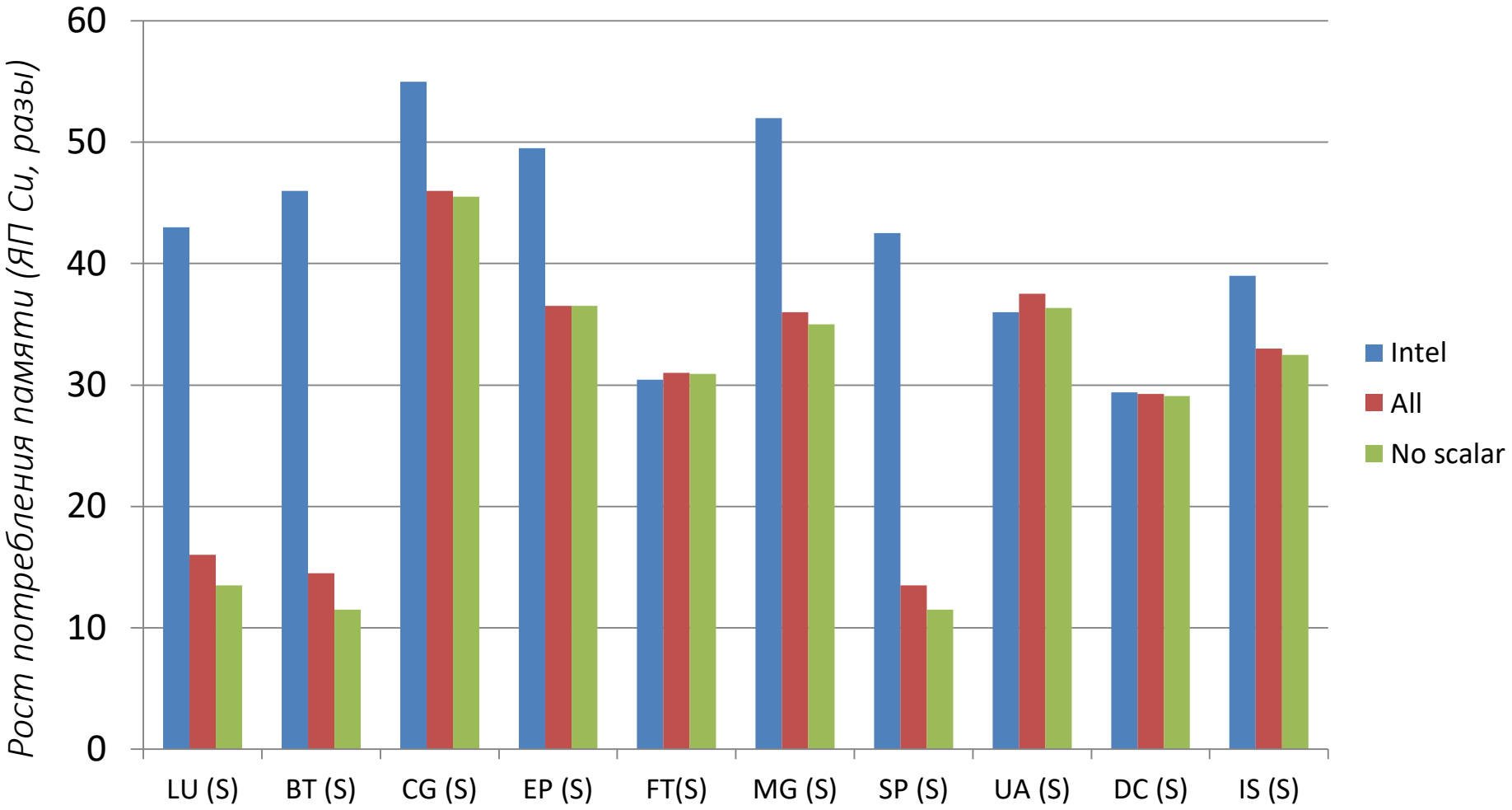
Анализ тестов NAS Parallel Benchmarks 3.3.1

Замедление при анализе трети всех циклов средствами Intel Advisor ~5000 раз.
Замедление при анализе всех циклов средствами SAPFOR ~2000 раз.



Intel Xeon CPU E5-1660 v2, 3.70 GHz, с отключенным Turbo Boost.
Clang и Flang версий 7.1.0 (библиотеки GCC 7.4), компиляция с опцией –O3.
Intel Advisor 2019, компиляция с опцией –O0 (замедление относительно –O3).

Анализ тестов NAS Parallel Benchmarks 3.3.1



Intel Xeon CPU E5-1660 v2, 3.70 GHz, с отключенным Turbo Boost.
Clang и Flang версий 7.1.0 (библиотеки GCC 7.4), компиляция с опцией –O3.
Intel Advisor 2019, компиляция с опцией –O0 (замедление относительно –O3).

Заключение

Разработанный инструмент может быть использован, как для получения результатов анализа с целью автоматизации распараллеливания программ в системе SAPFOR, так и с целью ручного распараллеливания программ.

Динамический анализ в системе SAPFOR позволяет предоставить рекомендации по устранению зависимостей:

- за счет приватизации переменных;
- за счет организации конвейерного выполнения.

Совместное использование со статическим анализом позволяет снизить накладные расходы на проведение динамического анализа, не потеряв полноты получаемых результатов.

Текущие исследования направлены на дальнейшее снижение накладных расходов за счет выполнения выборочной инструментации программ на основе статического анализа агрегатных типов данных.

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