МЕЖДУНАРОДНАЯ КОНФЕРЕНЦИЯ
«ИВАННИКОВСКИЕ ЧТЕНИЯ»

Построение распределения данных и генерация кода при распараллеливании на гетерогенный вычислительный кластер

Н.А. Катаев, А.С. Колганов

Институт прикладной математики им. М.В. Келдыша РАН

23 сентября, 2022 | Казань

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).

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.

CDVMH программа с распределением данных

```
#pragma dvm array distribute[block][block]
float A[L][L];
int main(int argc, char *argv[]) {
  float MAXEPS = 0.5E-5f;
  float w = 0.5f;
  for(int it = 0; it < ITMAX; it++) {
    float eps = 0.f;
    #pragma dvm actual(eps)
    #pragma dvm region
    {
      #pragma dvm parallel([i][j] on A[i][j]) \
                  across(A[1:1][1:1]), reduction(max(eps))
      for (int i = 1; i < L - 1; i++)
        for (int j = 1; j < L - 1; j++) {
          float s = A[i][j];
          A[i][j] = (A[i - 1][j] + A[i + 1][j] +
                     A[i][j - 1] + A[i][j + 1]) / 4.f;
          eps = Max(fabs(s - A[i][j]), eps);
        }
    }
    #pragma dvm get_actual(eps)
    printf("it=%4i   eps=%e\n", it, eps);
    if (eps < MAXEPS) break;
  }
  return 0;
}
```

Основные директивы:

- распределение данных,
- вычислительные области и спецификации перемещения данных между CPU и GPU,
- распределение вычислений,
- свойства переменных и спецификации доступа к удаленным данным.

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.

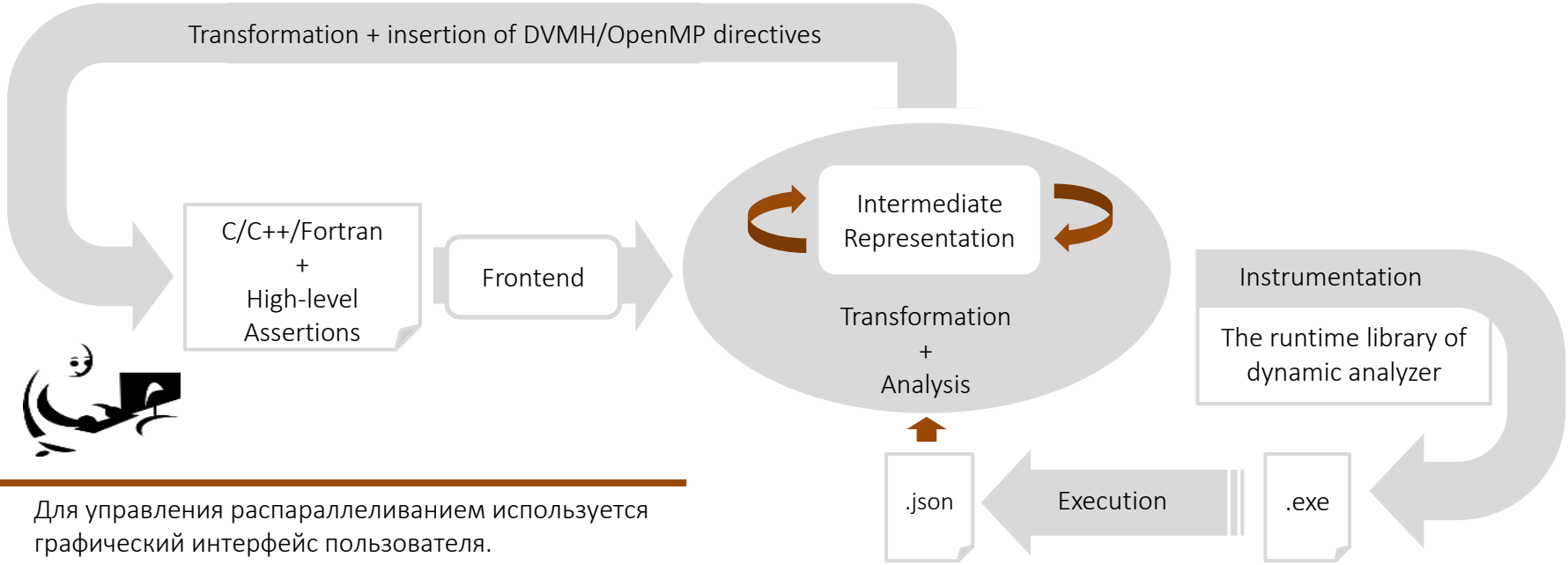
Система SAPFOR

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.

Автоматическое распараллеливание в DVMH модели

Для распараллеливание на гетерогенный вычислительный кластер требуется, чтобы в исходный код были вставлены следующие виды директив:

- *спецификации распределения данных (distribute, align)*,
- спецификации для циклов, которые могут выполняться параллельно: *спецификации распределения вычислений (parallel, on)*, спецификации приватных и редукционных переменных, а также шаблон доступа к массиву,
- *спецификации доступа к удаленным данным (shadow, shadow_renew, remote_access)*,
- спецификация вычислительных областей, которые могут быть выполнены на ускорителях, каждая область может содержать один или несколько параллельных циклов,
- высокоуровневые спецификации передачи данных между памятью центрального процессора и памятью ускорителя (директивы актуализации данных).

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.

Распределение данных

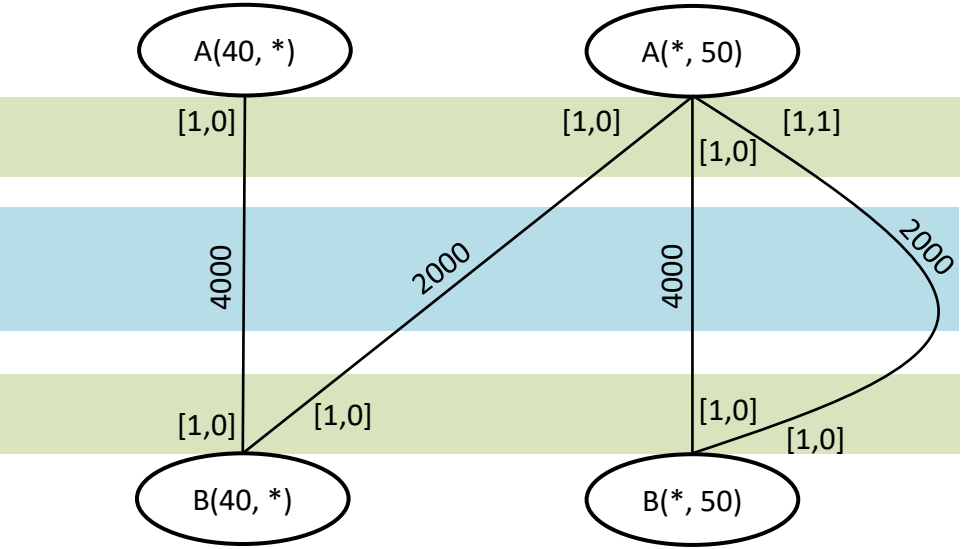
Модель выполнения программы

- Вычислительные узлы задают виртуальную многомерную решетку процессоров.
- Правило собственных вычислений, определяет, где должен быть выполнен каждый оператор присваивания.
- Возможен доступ к удаленным данным за счет механизма передачи сообщений.
- Источником параллелизма являются гнезда циклов, выполняющие вычисления над многомерными массивами данных.
- Элементы массивов, используемые на разных итерациях цикла, являются кандидатами для размещения на разных процессорах (распределение данных).
- Элементы массивов, используемые на одной итерации цикла, являются кандидатами для размещения на одном процессоре (выравнивание данных).

Распределение данных

- Выбрать распределяемые массивы (исключаются приватизируемые и редукционные массивы, массивы участвующие в операциях ввода-вывода, также пользователь может вручную запретить распределение некоторых массивов).
- Собрать информацию об обращениях к массивам во всех циклах программы, влияющих на выравнивание измерений массивов друг на друга (рассматриваются выражения вида a * i + b, остальные выражения порождают дополнительные обмены данными).
- Оценить коммуникационные издержки, возникающие при нарушении каждого из возможных правил выравнивания.
- Выбрать совокупность правил выравнивания данных, порождающую наименьшие коммуникационные издержки.

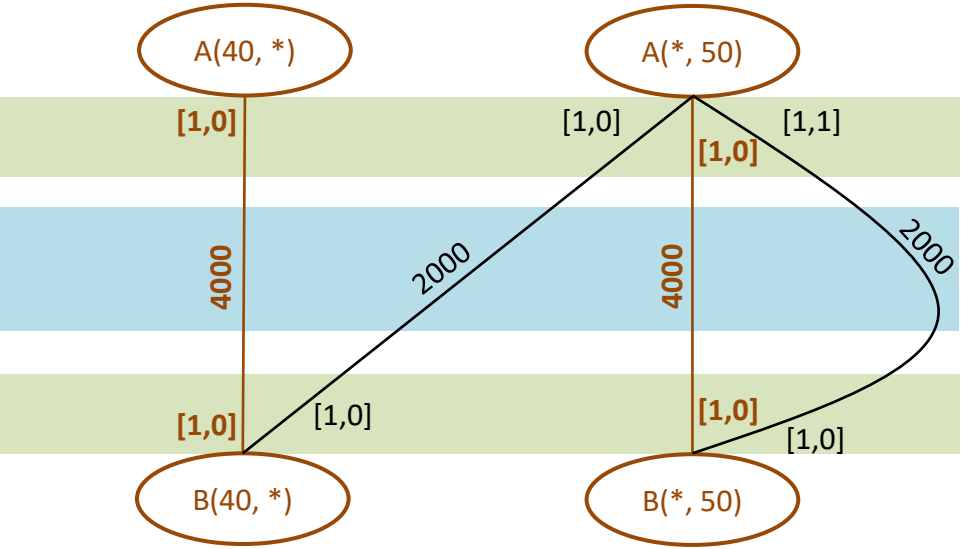
Граф массивов




```
int A[40][50], B[40][50];

void foo(int N, int M) {
  for (int I = 0; I < N; ++I) {
    for (int K = 0; K < M; ++K)
      A[I][K] = B[I][K] + B[K][K];
  }
  for (int I = 0; I < N; ++I) {
    for (int K = 0; K < M -1; ++K)
      A[I][K] = B[I][K] + B[I][K + 1];
  }
}
```

- *Вершина графа* – измерение распределяемого массива.
- *Ребро графа* – рекомендация по выравниванию, порожденная обращениями к массивам в программе.
- *Вес ребра* – оценка коммуникационных издержек в случае невыполнения соответствующего правила выравнивания.

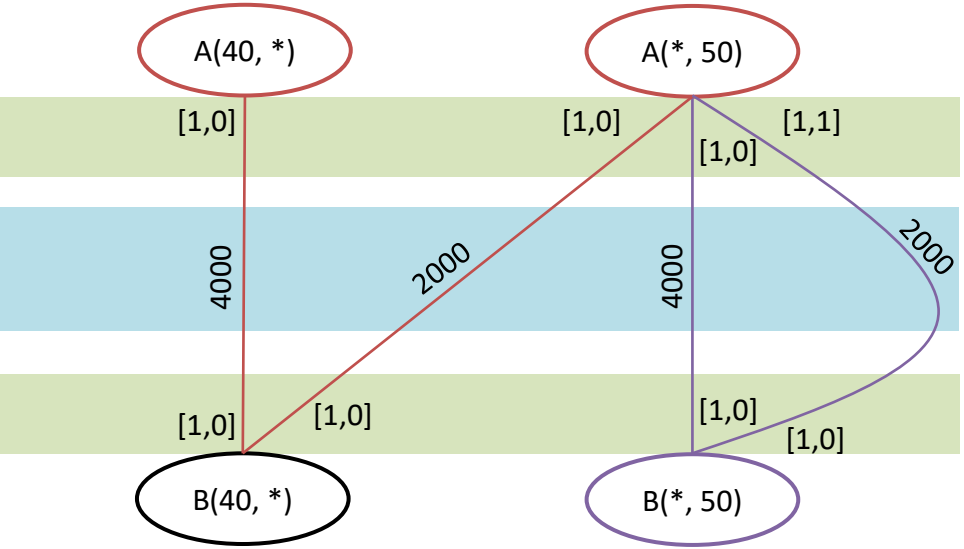
Граф массивов




```
int A[40][50], B[40][50];

void foo(int N, int M) {
  for (int I = 0; I < N; ++I) {
    for (int K = 0; K < M; ++K)
      A[I][K] = B[I][K] + B[K][K];
  }
  for (int I = 0; I < N; ++I) {
    for (int K = 0; K < M - 1; ++K)
      A[I][K] = B[I][K] + B[I][K + 1];
  }
}
```

- Обращения к массивам в циклах программы с разными индексными выражениями порождают разные ребра в графе.

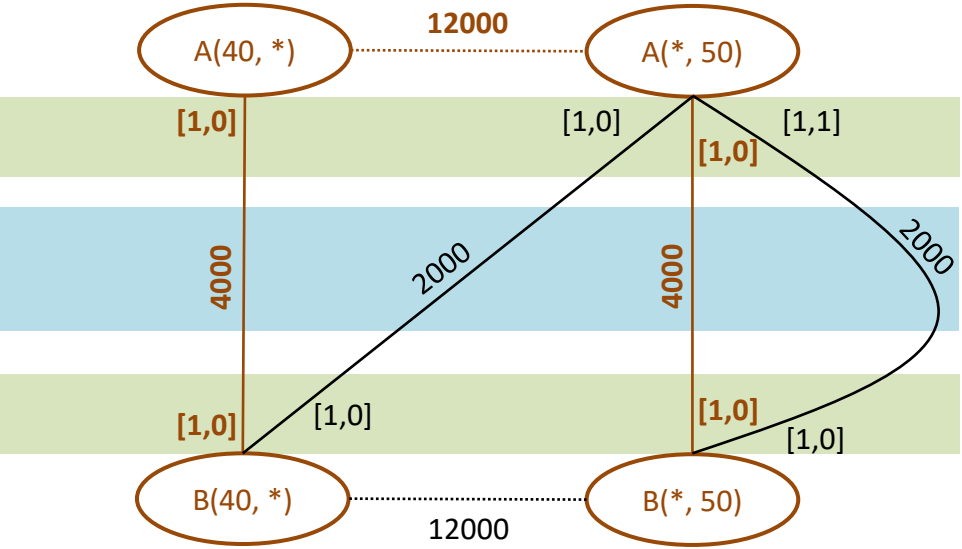
Конфликты выравнивания массивов




```
int A[40][50], B[40][50];

void foo(int N, int M) {
  for (int I = 0; I < N; ++I) {
    for (int K = 0; K < M; ++K)
      A[I][K] = B[I][K] + B[K][K];
  }
  for (int I = 0; I < N; ++I) {
    for (int K = 0; K < M - 1; ++K)
      A[I][K] = B[I][K] + B[I][K + 1];
  }
}
```

- Выравнивание разных измерений массива друг на друга.
- Неоднозначное выравнивание двух разных массивов.

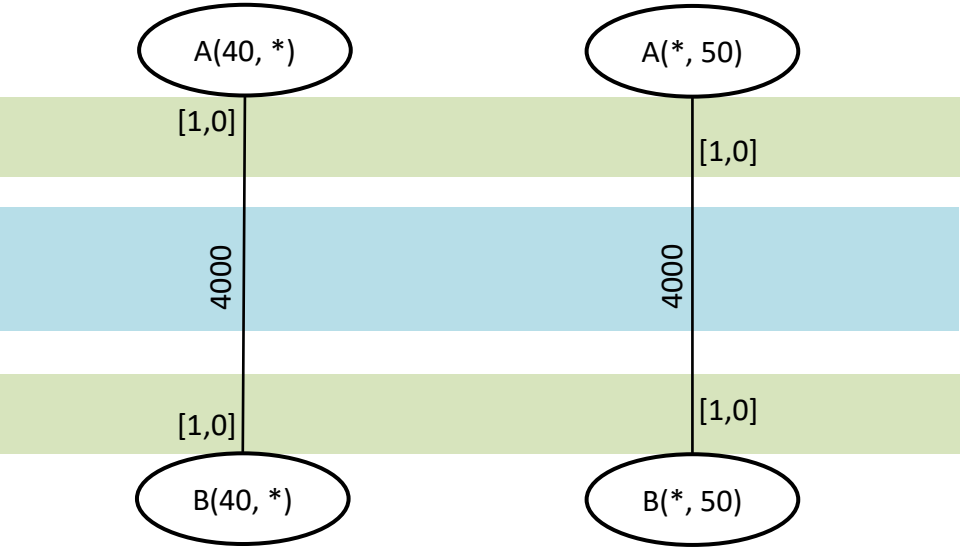
Максимальное остовное дерево




```
int A[40][50], B[40][50];

void foo(int N, int M) {
  for (int I = 0; I < N; ++I) {
    for (int K = 0; K < M; ++K)
      A[I][K] = B[I][K] + B[K][K];
  }
  for (int I = 0; I < N; ++I) {
    for (int K = 0; K < M - 1; ++K)
      A[I][K] = B[I][K] + B[I][K + 1];
  }
}
```

- Необходимо построить подграф без конфликтов выравнивания массивов, минимизирующий коммуникационные издержки.

Усеченный граф массивов




```
#pragma dvm array distribute[block][block]
int A[40][50];
#pragma dvm array align([I][J] with A[I][J])
int B[40][50];
```

```
int A[40][50], B[40][50];

void foo(int N, int M) {
  for (int I = 0; I < N; ++I) {
    for (int K = 0; K < M; ++K)
      A[I][K] = B[I][K] + B[K][K];
  }
  for (int I = 0; I < N; ++I) {
    for (int K = 0; K < M - 1; ++K)
      A[I][K] = B[I][K] + B[I][K + 1];
  }
}
```

Исследование программ из NAS Parallel Benchmarks 3.3.1

Инлайн подстановка функций на уровне внутреннего представления программы, чтобы убедиться в отсутствии зависимостей по данным (BT).

Динамический анализ приватизируемых массивов (BT, EP).

Использование опций анализа:

- чтобы указать, что выражение индекса не выходит за пределы выделенной памяти (BT),
- чтобы игнорировать возможность побочного эффекта математических функций, сохраняемого в переменной *errno* (EP).

Ручное преобразование приложений EP и BT

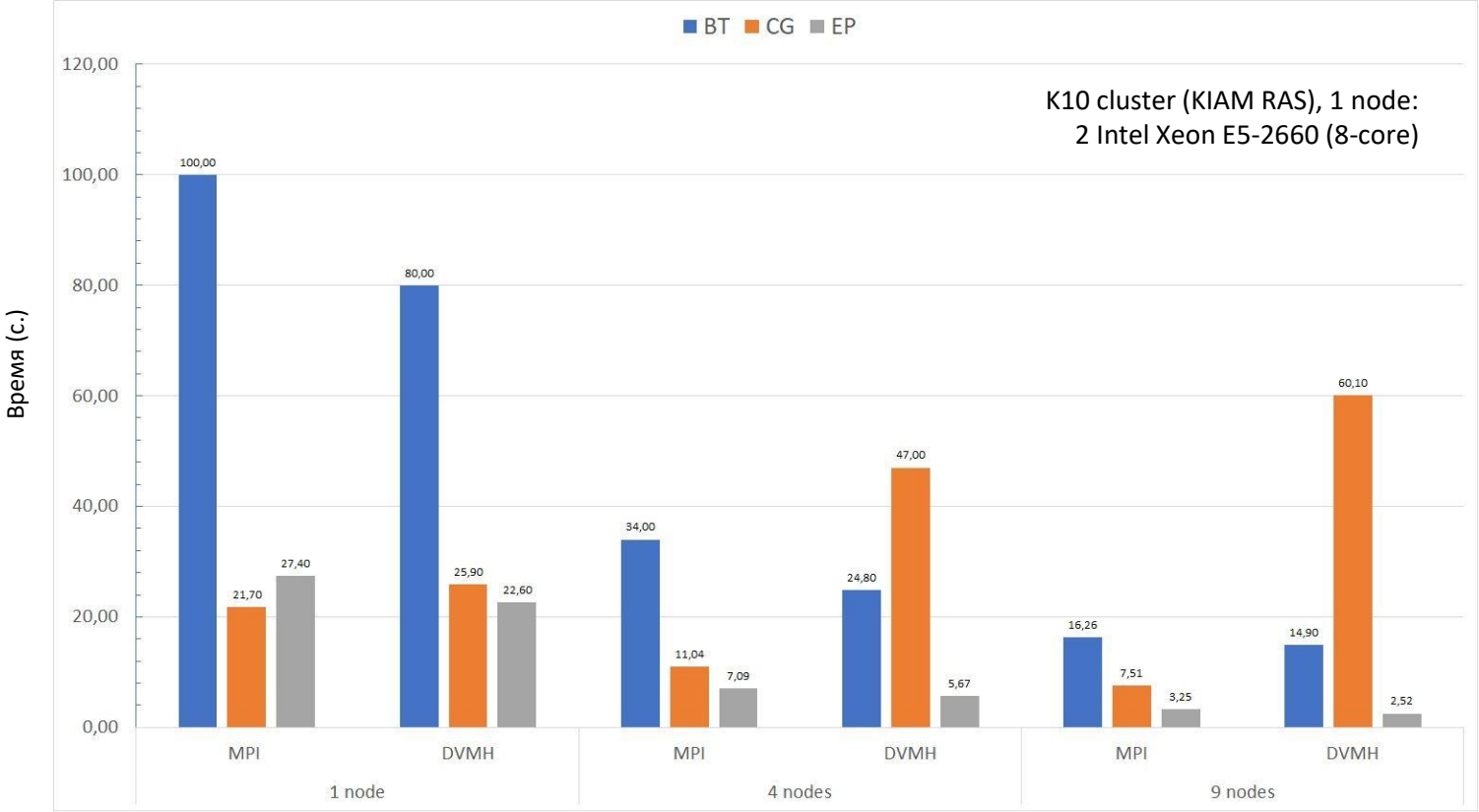
Устранение большого приватизируемого массива для уменьшения использования памяти на GPU (EP, BT):

```
for (i = 0; i < NK; i++) {
  ...
  x[i] = r46 * (*x4);
}

for (i = 0; i < NK; i++) {
  x1 = 2.0 * x[2 * i] - 1.0;
  x2 = 2.0 * x[2 * i + 1] - 1.0;
  ...
}
```

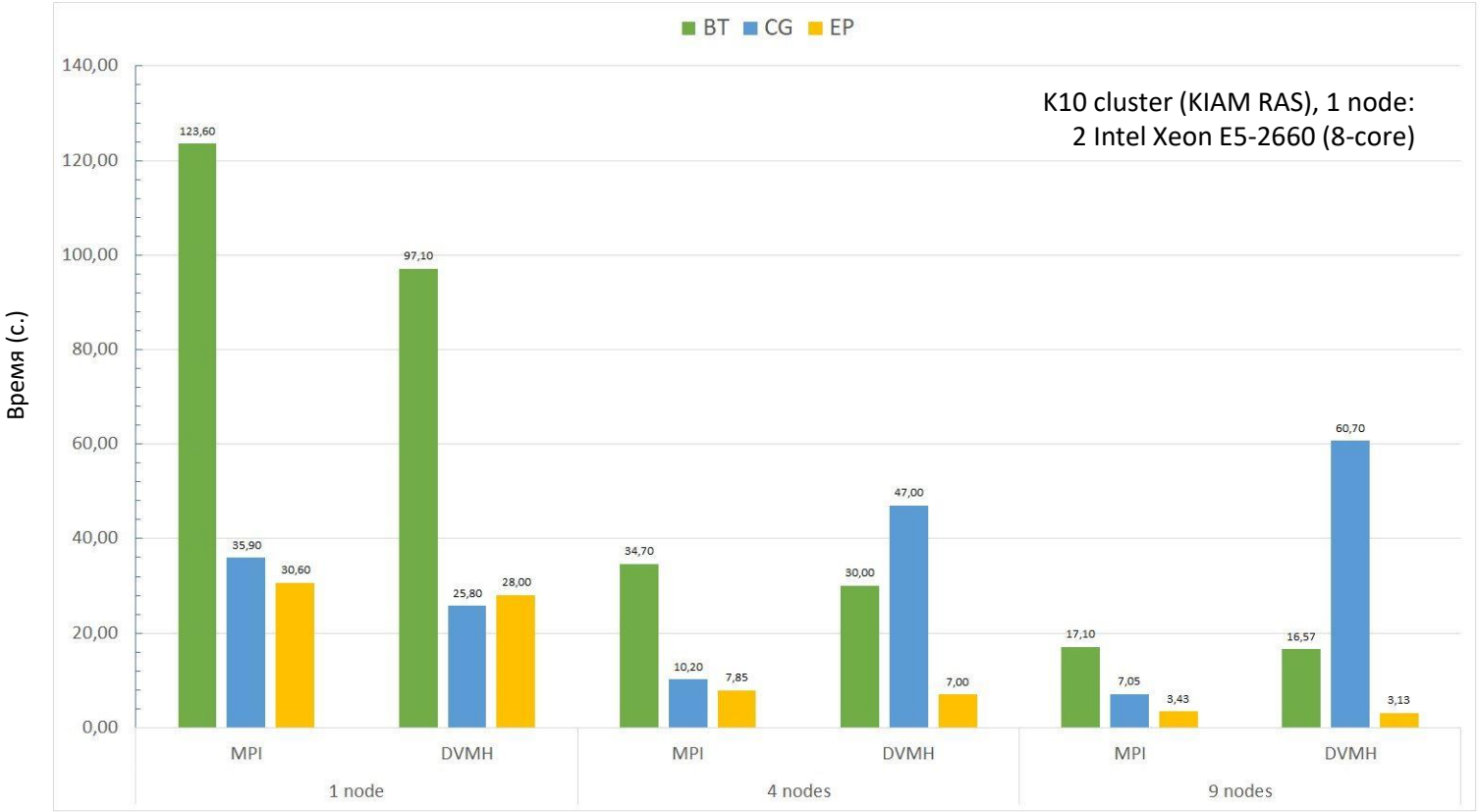
⇨

```
for (i = 0; i < NK; i++) {
  double x_2i, x_2i1;
  ...
  x_2i = r46 * (*x4);
  ...
  x_2i1 = r46 * (*x4);

  x1 = 2.0 * x_2i - 1.0;
  x2 = 2.0 * x_2i1 - 1.0;
  ...
}
```

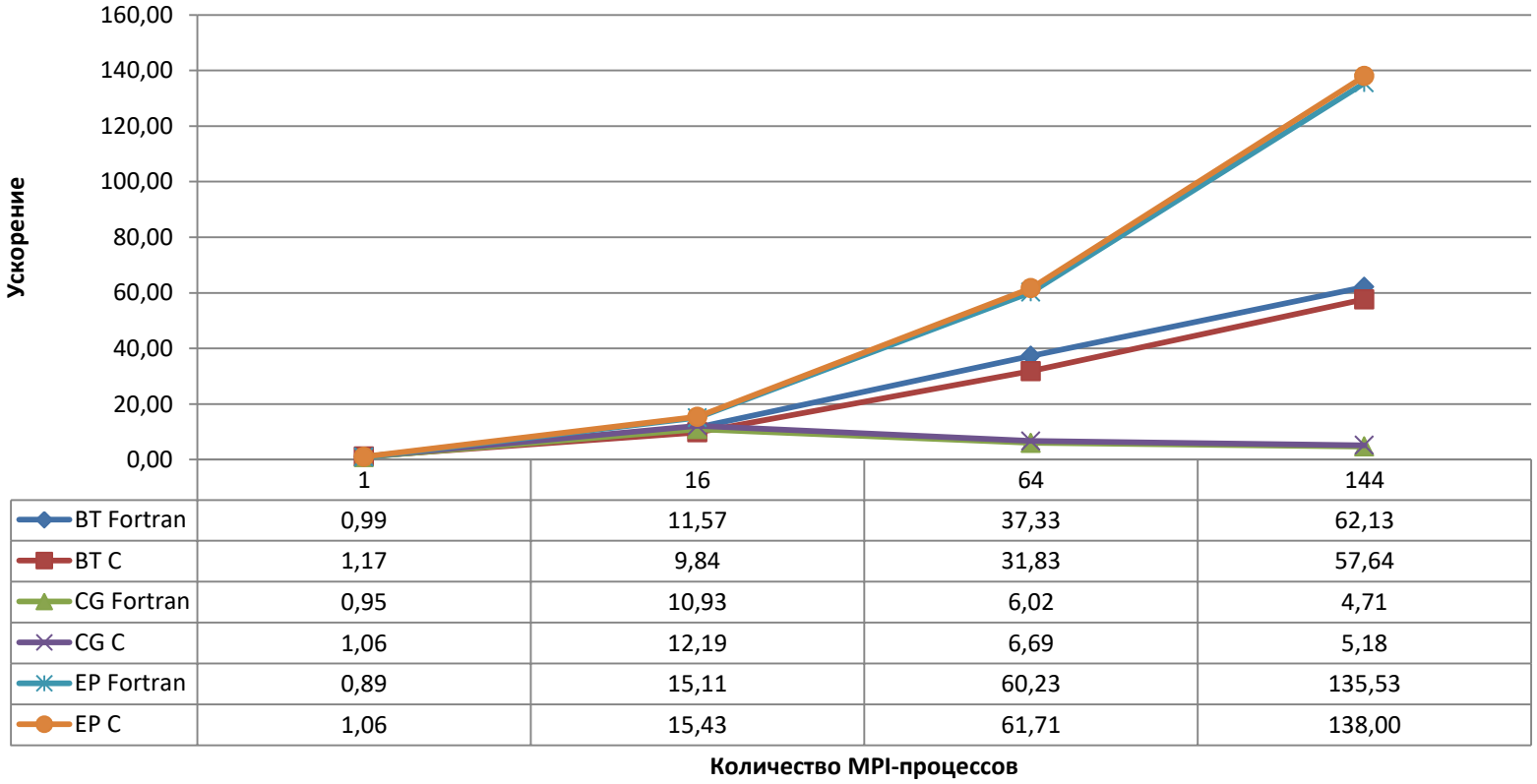
Распараллеливание NAS Parallel Benchmarks 3.3.1 (Fortran)



Распараллеливание NAS Parallel Benchmarks 3.3.1 (C)



Ускорение DVMH версий NAS Parallel Benchmarks 3.3.1



	1	16	64	144
BT Fortran	0,99	11,57	37,33	62,13
BT C	1,17	9,84	31,83	57,64
CG Fortran	0,95	10,93	6,02	4,71
CG C	1,06	12,19	6,69	5,18
EP Fortran	0,89	15,11	60,23	135,53
EP C	1,06	15,43	61,71	138,00

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.

Автоматическое распараллеливание на вычислительный кластер в системе SAPFOR охватывает решение всех трех основных задач:

- распределение данных,
- распределение вычислений,
- организация доступа к удаленным данным.

Для повышения производительности параллельных программ система SAPFOR полагается на различные оптимизации, реализованные в компиляторе и в runtime системе DVMH.

Системы SAPFOR и DVM, дополняя друг друга, могут значительно снизить затраты на разработку параллельных программ для гетерогенных вычислительных кластеров.

Спасибо за внимание!

http://dvm-system.org

dvm@keldysh.ru

МЕЖДУНАРОДНАЯ КОНФЕРЕНЦИЯ
«ИВАННИКОВСКИЕ ЧТЕНИЯ»