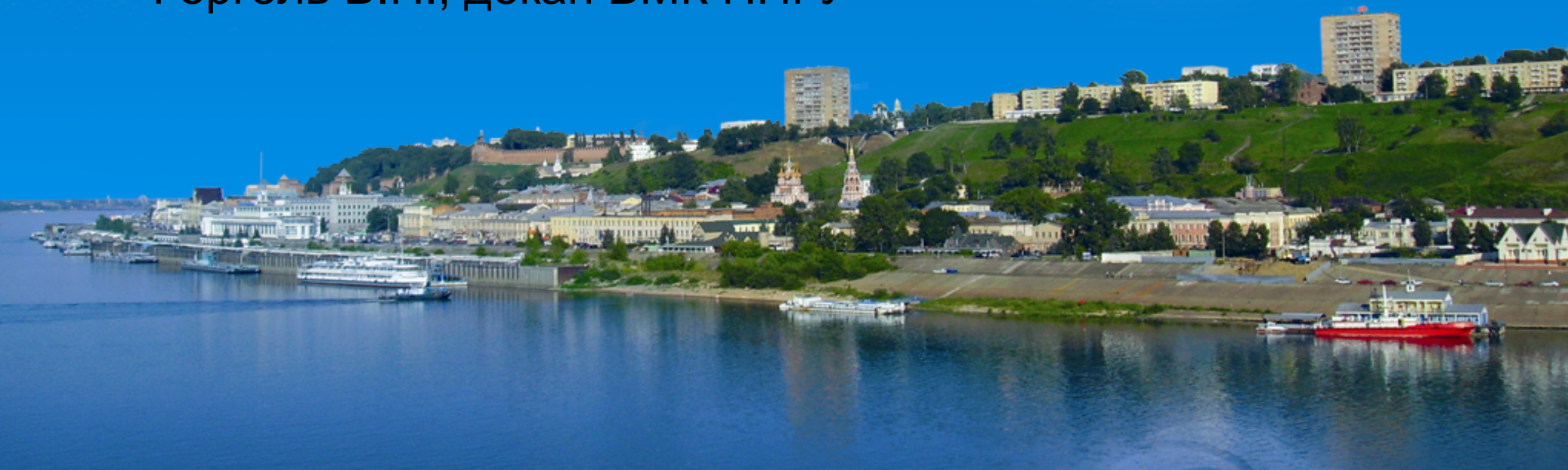
Нижегородский государственный университет
им. Н.И. Лобачевского
- Национальный исследовательский университет -

Лекция.

Язык параллельного программирования Co-Array Fortran

Гергель В.П., декан ВМК ННГУ



Содержание

- Язык параллельного программирования Co-Array Fortran
  - Подходы к разработке параллельных программ
  - Модель параллельного программирования CAF
  - Понятие исполнителей (images)
  - Понятие распределенных массивов (co-arrays)
  - Синхронизация параллельных вычислений
  - Примеры
  - Оценка эффективности CAF-программ
  - Развитие: от CAF 1.0 к CAF 2.0

Подходы к разработке параллельных программ

- **Использование библиотек** для существующих языков программирования – MPI для C и Fortran для систем с распределенной памятью
- **Использование надъязыковых средств** (директив, комментариев) - OpenMP для C и Fortran для систем с общей памятью
- **Расширение существующих языков программирования** – например, UPC, **CAF**
- **Создание новых – параллельных – языков программирования** – например, Chapel, X10, Fortress,…

CAF: Введение…

- Организация вычислений в рамках модели PGAS (***разделенное глобально-адресуемое пространство***)
  - Двухуровневое представление памяти для управления локальностью (локальная/удаленная память),
  - Управление распределением данных в локальной памяти,
  - Контроль за передачами данных из удаленных областей памяти.
- CAF (Co-Array Fortran) был предложен Numrich и Reid (университет Миннесоты, США) в середине 90-х. Начальное названия языка – F--.
- Разработка языка поддержана компанией Cray. Один из наиболее активных центров по развитию CAF – Rice University.

Основа подхода *– минимальное (но продуктивное) расширение языка Fortran для превращения его эффективный язык параллельного программирования*

CAF: Введение

- Пример программы

```
program Hello_World
  implicit none
  integer :: i ! локальная переменная
  character(len=20) :: name[*]! co-array переменная
  if (this_image() == 1) then
    print *,'Enter your name: '
    read (*,'(a)') name

    ! рассылка данных
    do i = 2, num_images()
      name[i] = name
    end do
  end if

  sync all ! дожидаемся всех images

  print *, 'Hello ' , name, 'from image ', this_image()
end program Hello_world
```

CAF: Модель программирования…

- **Основа – схема SPMD «Single-Program-Multiple-Data» :**
  - Разрабатывается один экземпляр CAF-программы, который далее копируется необходимое количество раз. Копии CAF-программы могут выполняться параллельно,
  - Каждая копия программы содержит свои «локальные» данные,
  - Данные, к которым требуется доступ из разных копий CAF-программы, должны описываться особым образом (*co-arrays*). Передача данных между копиями программы осуществляется только при помощи явно указываемых действий.

CAF: Модель программирования

- **Копия CAF-программы в терминологии CAF называется *image*:**
  - Image может рассматриваться как некоторая абстракция вычислительного элемента и будет именоваться далее ***Исполнитель***,
  - Количество созданных исполнителей может быть определено при помощи функции **`num_images()`**,
  - Количество исполнителей, как правило, соответствует числу имеющихся процессоров; в общем же случае, количество исполнителей может отличаться от числа процессоров,
  - Каждый исполнитель характеризуется своим уникальным индексом (функция **`this_image()`**).

Индексы могут быть использованы для разделения вычислений в исполнителях (по аналогии с MPI-программированием)

CAF: Распределенные массивы…

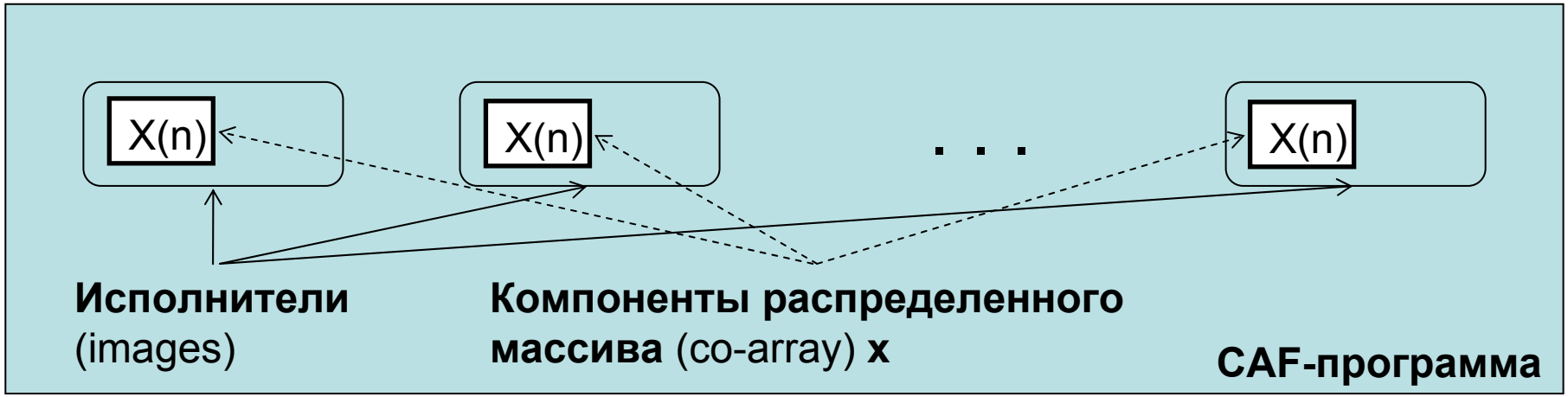
- **Для работы с данными, распределенными между исполнителями, в CAF вводится механизм *co-arrays*.**
- **Синтаксис**

  co-array: `<type> :: <data_declaration> [*]`
- **Семантика**

  Копирование определяемого объекта данных (массива) по всем исполнителям CAF-программы:
  - Номер копии распределенного массива (индекс исполнителя) указывается в квадратных скобках (индекс обычного массива задается в круглых скобках), т.е. скобки () обеспечивают доступ к локальной памяти, скобки [] – доступ к удаленной памяти,
  - Номер копии, локальной для исполнителя, может не приводиться.
- **Пример**
  - real :: x(n)[*] ! Создание копии массива **x** на всех исполнителях программы

CAF: Распределенные массивы…

- **Пример**
  - real :: x(n)[*] ! Создание копии массива **x** на всех исполнителях программы



- **Работа с компонентами распределенного массива выполняется в соответствии с правилами Fortran**
  - **x(1)[p] = x(n)[q]** ! Чтение элемента исполнителя *q* и запись элемента для исполнителя *p*

CAF: Распределенные массивы…

- **Определение распределенного массива может выполняться в полном соответствии с правилами Fortran:**

```
real :: a(n)[*]
complex :: z[0:*]
integer :: index(n)[*]
real :: b(n)[p, *]
real :: c(n,m)[0:p, -7:q, +11:*]
real, allocatable :: w(:)[:]
type(field) :: maxwell[p,*]
```

CAF: Распределенные массивы…

- **Пример**

```
int :: x[4,*]
num_image()   = 16
this_image()  = 15
this_image(x) = (/3,4/)
```

	1	2	3	4
1	**1**	**5**	**9**	**13**
2	2	6	10	14
3	3	7	11	*15*
4	4	8	12	16

- Максимальное значение индекса по второму измерению определяется как
  `_ = num_image()/4`
- Первый индекс может трактоваться, например, как номер ядра в процессоре

CAF: Распределенные массивы…

- **Пример**

```
int :: x[0:3,0:*]
num_image()   = 16
this_image() = 15
this_image(x) = (/2,3/)
```

	0	1	2	3
0	**1**	**5**	**9**	**13**
1	2	6	10	14
2	3	7	11	*15*
3	4	8	12	16

CAF: Распределенные массивы…

Пример

```
int :: x[-5:-2,0:*]
num_image()   = 16
this_image()  = 15
this_image(x) = (/-3,3/)
```

	0	1	2	3
-5	**1**	**5**	**9**	**13**
-4	2	6	10	14
-3	3	7	11	*15*
-2	4	8	12	16

CAF: Распределенные массивы…

- **Пример**

```
int :: x[-5:-2,0:*]
num_image() = 14
```

	0	1	2	3
-5	**1**	**5**	**9**	**13**
-4	2	6	10	14
-3	3	7	11	-
-2	4	8	12	-

- Исполнители с индексами /-3,3/ и /-2,3/ не определены

CAF: Распределенные массивы…

- **Пример**

```
int :: x[p,q,*]
```

  - Организация набора исполнителей в виде трехмерной решетки
  - Максимальное значение индекса по третьему измерению определяется как
  - `= num_image()/(p*q)`

CAF: Распределенные массивы

- **Работа с компонентами распределенного массива также выполняется в полном соответствии с правилами Fortran:**

```
y(:) = x(:)[p]
x(index(k)) = y[index(p)]
x(:)[q] = x(:) + x(:)[p]
u(2:n-1)[p] = u(1:n-2)[q] + u(3:n)[r]
```

- Отсутствие индекса компонента распределенного массива означает доступ к компоненте, локальной для исполнителя

CAF: Синхронизация…

Барьерная синхронизация…

- **Синтаксис**

```
sync all
```

- **Семантика**

Барьерная синхронизация – исполнители приостанавливаются на барьере до тех пор, пока на барьере не окажутся все исполнители, т.е. действия любого исполнителя до барьера будут выполнены ранее действий любого другого исполнителя после барьера.

CAF: Синхронизация…

Барьерная синхронизация

- **Пример –** чтение в исполнителе 1 и рассылка всем

```
real :: z[*]
  ...
sync all
if (this_image()==1) then
    read(*,*) z
    do image = 2, num_images()
      z[image] = z
    end do
end if
sync all
...
```

CAF: Синхронизация…

Синхронизация групп исполнителей…

- **Синтаксис**

  sync images (<image-set>)

- **Семантика**
  - Синхронизация исполнителей, указываемых в качестве параметра функции; параметр **<image-set>** есть массив индексов исполнителей, для которых должны быть выполнена синхронизация (для указания всех исполнителей можно указать символ *)
  - Действие sync images (*) не является эквивалентным с sync all – для sync images не требуется использование одного и того значения параметра для всех исполнителей, участвующих в синхронизации
    (см. пример)

CAF: Синхронизация…

Синхронизация групп исполнителей

- **Пример –** Синхронизация исполнителя 1 со всеми другими исполнителями (но не каждого с каждым)

```
if (this_image() == 1) then
  ! Задание данных для всех исполнителей
  sync images(*)
else
  sync images(1)
  ! Использование данных
end if
```

CAF: Синхронизация…

Синхронизация критических секций

- Синтаксис

```
critical
   ! Код, выполняемый в любой момент времени
   ! только одним исполнителем
end critical
```

CAF: Синхронизация

Синхронизация памяти

- **Синтаксис**

  `sync memory`

- **Семантика**

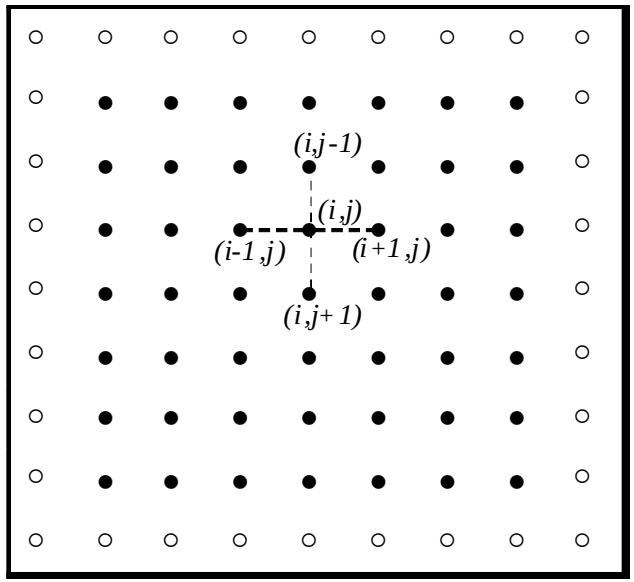
  Синхронизация временной и основной памяти
  (сохранение данных из временной памяти)

CAF: Пример 1 – Метод конечных разностей…

- Задача Дирихле с периодическими граничными условиями
- Метод конечных разностей – пятиточечный шаблон
- Организация данных – u(1:nrow)[1:ncol]

Метод Гаусса-Зейделя

$$u_{ij}^{k} = u_{i-1,j}^{k} + u_{i+1,j}^{k-1} + u_{i,j-1}^{k} + u_{i,j+1}^{k-1} - 4h^2 f_{ij}$$



CAF: Пример 1 – Метод конечных разностей…

```
subroutine laplace (nrow,ncol,u)
  integer, intent(in)   ::  nrow, ncol
  real, intent(inout)   ::  u(nrow)[*]
  real                  ::  new_u(nrow)
  integer               ::  i, me, left, right
  new_u(1) = u(nrow) + u(2)                                          (1)
  new_u(nrow) = u(1) + u(nrow-1)                                     (2)
  new_u(2:nrow-1) = u(1:nrow-2) + u(3:nrow)
  me = this_image(u) ! Returns the co-subscript within u
                     ! that refers to the current image
  left = me-1; if (me == 1) left = ncol
  right = me + 1; if (me == ncol) right = 1
  sync all( (/left,right/) )  ! Wait if left and right               (3)
                              ! have not already reached here
  new_u(1:nrow)=new_u(1:nrow)+u(1:nrow)[left]+u(1:nrow)[right]       (4)
  sync all( (/left,right/) )
  u(1:nrow) = new_u(1:nrow) - 4.0*u(1:nrow)
end subroutine laplace
```

CAF: Пример 1 – Метод конечных разностей

- **Пояснения к программе:**
  - (1) – учет периодичности граничных условий
  - (2) – прибавление значений соседних узлов по горизонтали
  - (3) – синхронизация готовности предшествующих вычислений
  - (4) - прибавление значений соседних узлов по вертикали

CAF: Пример 2 – Поиск максимума в распределенном массиве…

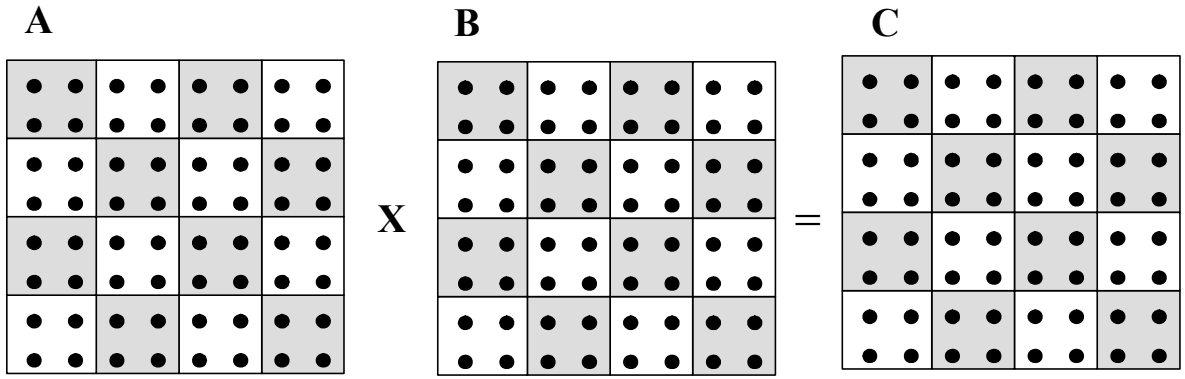
```
subroutine greatest(a,great) ! Find maximum value of a(:)[*]
  real, intent(in)  :: a(:)[*]
  real, intent(out) :: great[*]
  real :: work(num_images()) ! Local work array
  great = maxval(a(:))                                    (1)
  sync all ! Wait for all other images to reach here
  if(this_image(great)==1)then
    work(:) = great[:]     ! Gather local maxima          (2)
    great[:]=maxval(work) ! Broadcast the global maximum  (3)
  end if
  sync all
end subroutine greatest
```

CAF: Пример 2 – Поиск максимума в распределенном массиве

- **Пояснения к программе:**
  - (1) – нахождение локальных максимумов каждым исполнителем
  - (2) – `сбор локальных максимумов в исполнителе` 1
  - (3) – `нахождение общего максимума и рассылка по всем исполнителям`

CAF: Пример 3 – Матричное умножение…

- **Распределение данных** – Блочная схема



- **Базовая подзадача** - процедура вычисления всех элементов одного из блоков матрицы *C*

$$\begin{pmatrix} A_{00}A_{01}\dots A_{0q-1} \\ \dots \\ A_{q-10}A_{q-11}\dots A_{q-1q-1} \end{pmatrix} \times \begin{pmatrix} B_{00}B_{01}\dots B_{0q-1} \\ \dots \\ B_{q-10}B_{q-11}\dots B_{q-1q-1} \end{pmatrix} = \begin{pmatrix} C_{00}C_{01}\dots C_{0q-1} \\ \dots \\ c_{q-10}C_{q-11}\dots C_{q-1q-1} \end{pmatrix}, \quad C_{ij} = \sum_{s=1}^{q} A_{is}B_{sj}$$

CAF: Пример 3 – Матричное умножение…

Вариант 1

```
real,dimension(n,n)[p,*] :: a,b,c

! Вычисление блока матрицы C(myP,myQ)
! Размер блока n×n
! Исполнители – решетка p×p
do k = 1, n
  do q = 1, p
    c(i,j)[myP,myQ] = c(i,j)[myP,myQ]
                + a(i,k)[myP,q]*b(k,j)[q,myQ]
  enddo
enddo
```

CAF: Пример 3 – Матричное умножение

Вариант 2

```
real,dimension(n,n)[p,*] :: a,b,c
do k=1,n
  do q=1,p
    c(i,j) = c(i,j) + a(i,k)[myP, q]*b(k,j)[q,myQ]
  enddo
enddo
```

CAF: Оценка эффективности…

Оценка эффективности на основе теста NASA Parallel Benchmark (NPB)…

- Разработан в начале 90-х
- Разрабатывался как универсальное средство для оценки производительности суперкомпьютеров
- Включает в себя ядра задач гидро- и аэродинамического моделирования
- Официально представляет собой лишь набор правил и рекомендаций
- Готовая реализация теста доступна на сервере NASA

CAF: Оценка эффективности…

Оценка эффективности на основе теста NASA Parallel Benchmark (NPB)…

- EP — Embarrassing Parallel. Вычисление интеграла методом Монте-Карло
- MG — simple 3D MultiGrid benchmark. Приближенное решение трехмерного уравнения Пуассона ("трехмерная решетка") в частных производных на сетке NxNxN с периодическими граничными условиями
- CG — solving an unstructured sparse linear system by the Conjugate Gradient method. Вычисление наименьшего собственного значения больших, разреженных матриц методом сопряженных градиентов.
- FT —3-D Fast-Fourier Transform partial differential equation benchmark. Вычисление методом быстрого преобразования Фурье трехмерного уравнения в частных производных.
- IS — Parallel Sort of small Integers. Параллельная сортировка N целых чисел.
- LU — LU Solver. Тест выполняет решение системы уравнений с равномерно разреженной блочной треугольной матрицей методом симметричной последовательной верхней сверхрелаксации (symmetric successive over-relaxation — SSOR)
- SP — Scalar Pentadiagonal. Тест выполняет решение нескольких независимых систем скалярных уравнений
- BT — Block Tridiagonal. Решение серии независимых систем уравнений

Тест	Класс A	Класс B	Класс C	Класс D	Класс E
EP	2^{28}	2^{30}	2^{32}	2^{36}	2^{40}
MG	256^3	256^3	512^3	1024^3	2048^3
CG	14000	75000	$1.5x10^5$	$1.5x10^6$	$9x10^6$
FT	256^2x128	256^2x512	512^3	1024^2x2048	$4096x2048^2$
IS	2^{23}	2^{25}	2^{27}	2^{29}	
LU	64^3	102^3	162^3	408^3	1020^3
SP	64^3	102^3	162^3	408^3	1020^3
BT	64^3	102^3	162^3	408^3	1020^3
DT					

CAF: Оценка эффективности…

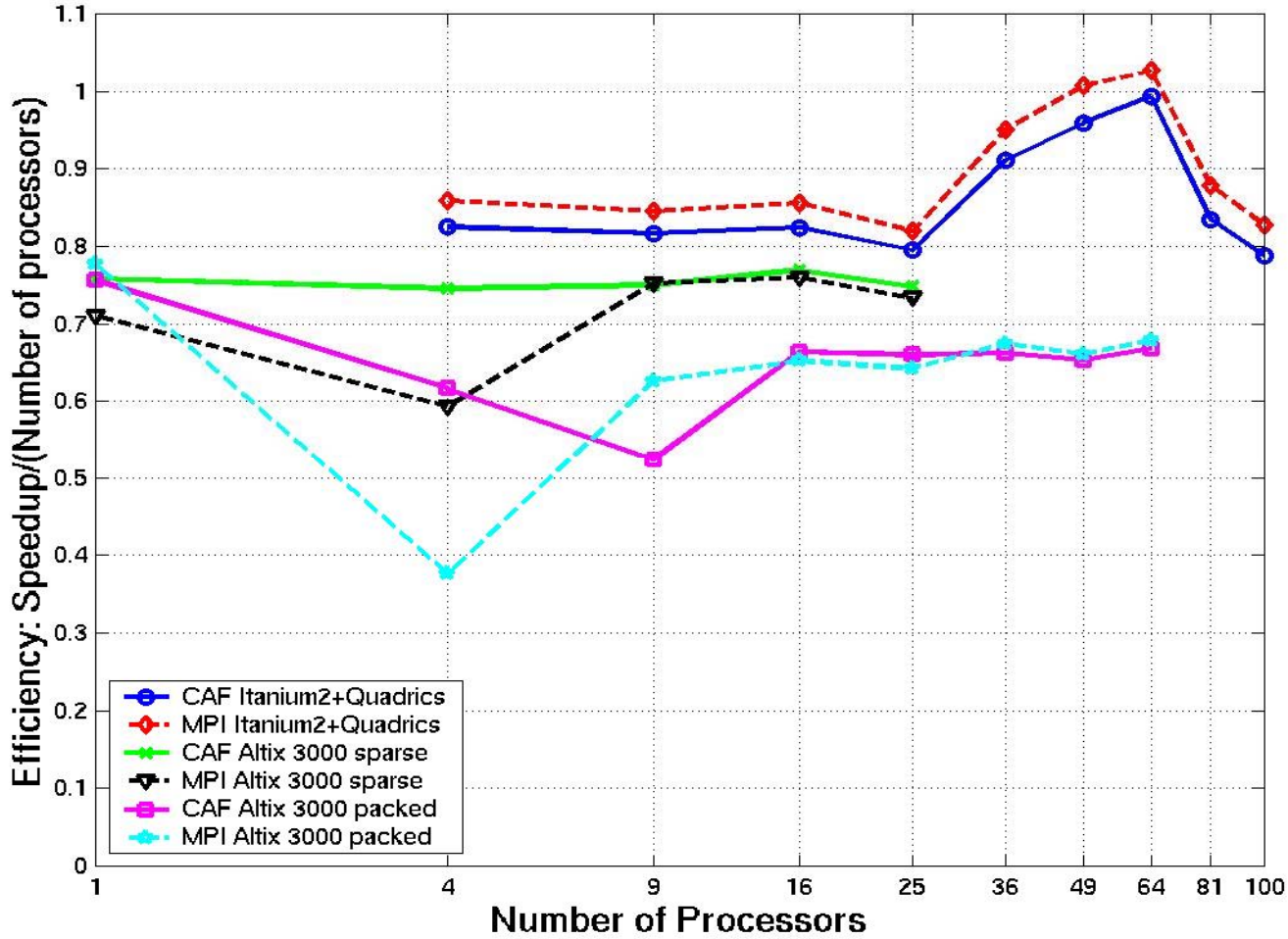
Оценка эффективности на основе теста NASA Parallel Benchmark (NPB)…

- NAS versions:
  - NPB2.3b2 : MPI реализация
  - NPB2.3-serial : Последовательный код на основе MPI варианта
- CAF version
  - NPB2.3-CAF: CAF реализация на основе MPI варианта
- Аппаратные платформы
  - SGI Altix 3000 (Itanium2 1.5GHz)
  - Itanium2+Quadrics QSNet II (Elan4, Itanium2 1.5GHz)

Результаты вычислительных экспериментов использованы из: Coarfa C., Dotsenko Yu. and Mellor-Crummey J. **Co-Array Fortran: Compilation, Performance, Language Issues.** SGI User Group Technical Conference (SGIUG 2004), Orlando, Florida, May 2004.

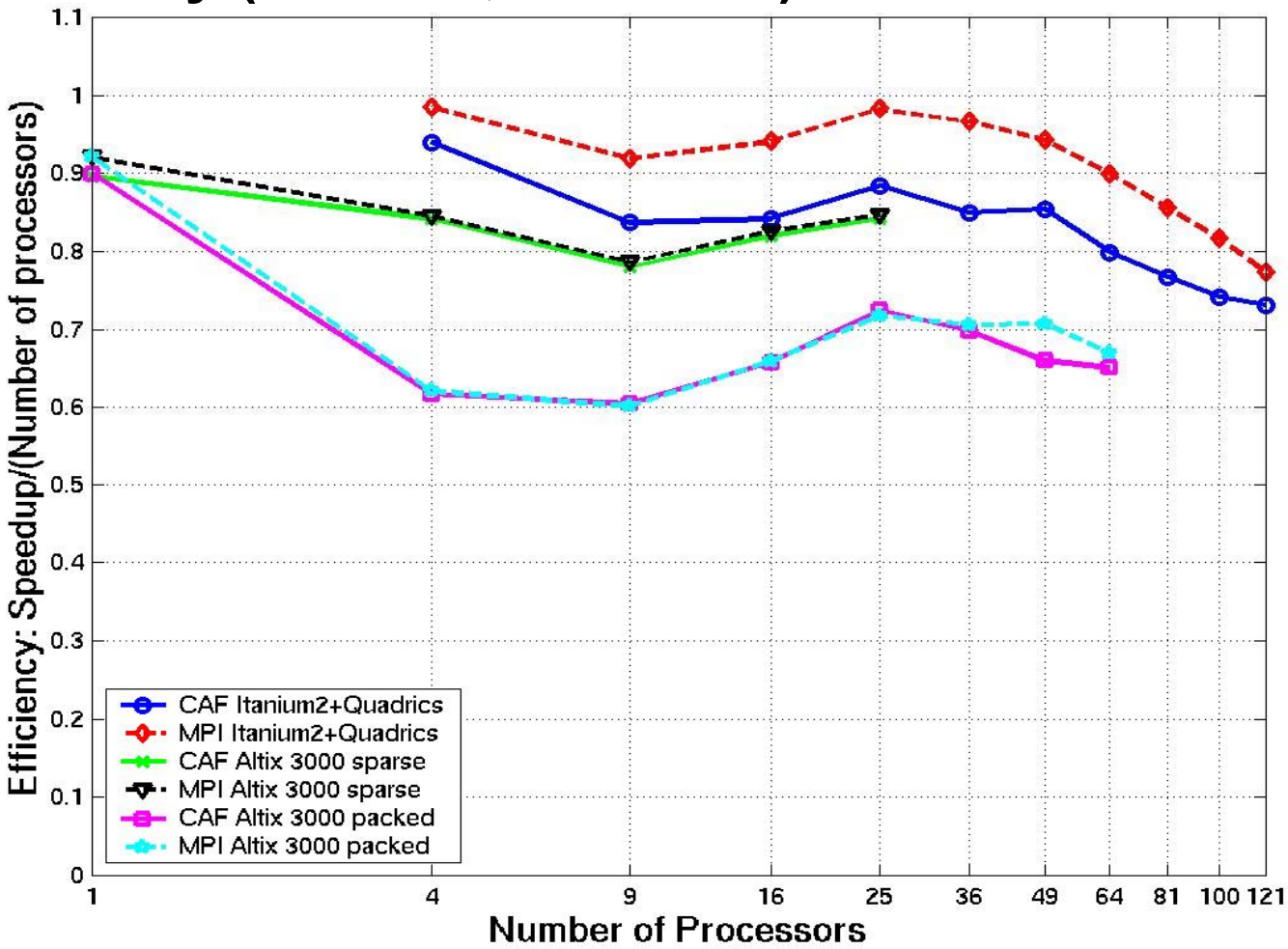
CAF: Оценка эффективности…

NPB BT Efficiency (Class C, size 162^3)



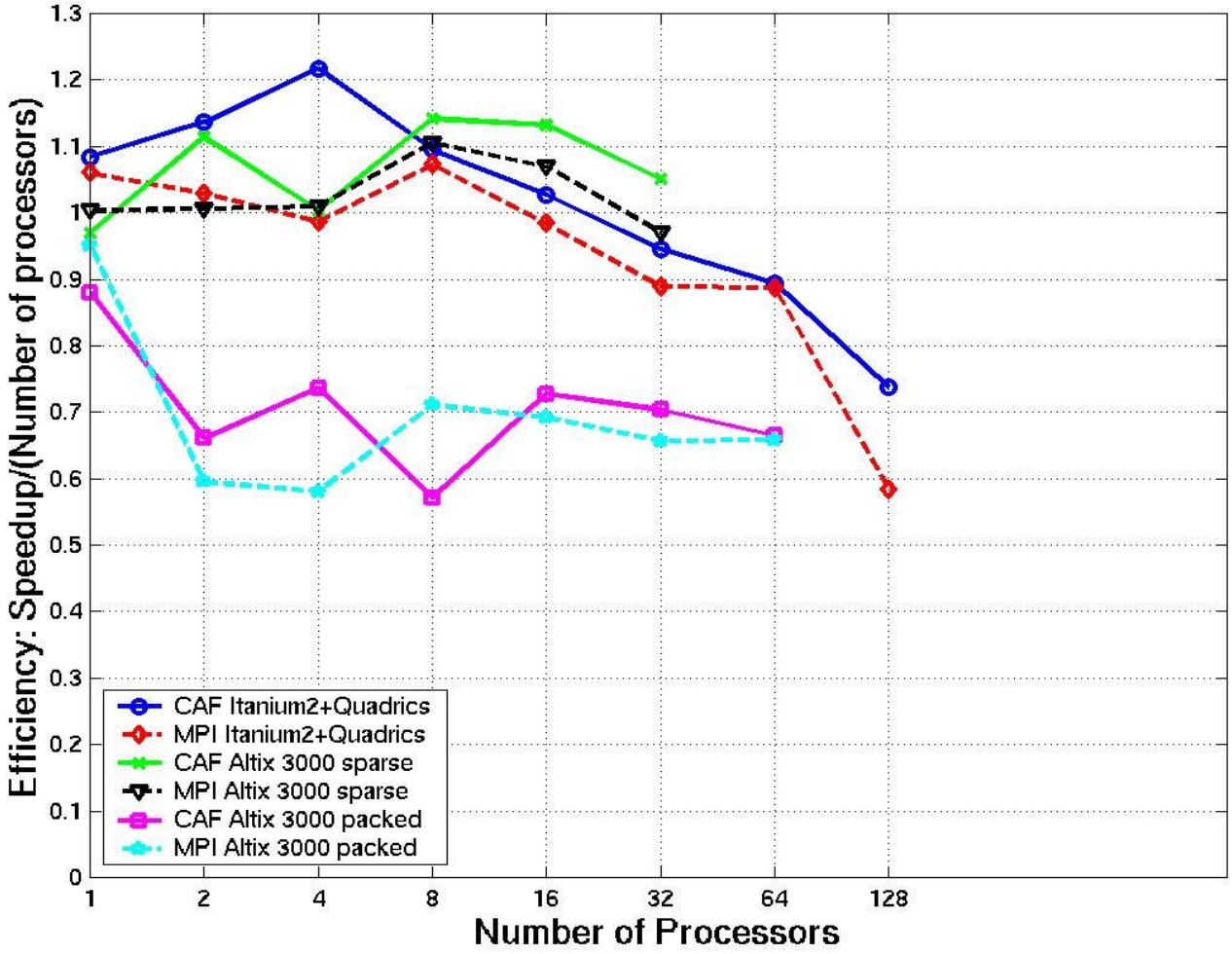
CAF: Оценка эффективности…

NPB SP Efficiency (Class C, size 162^3)



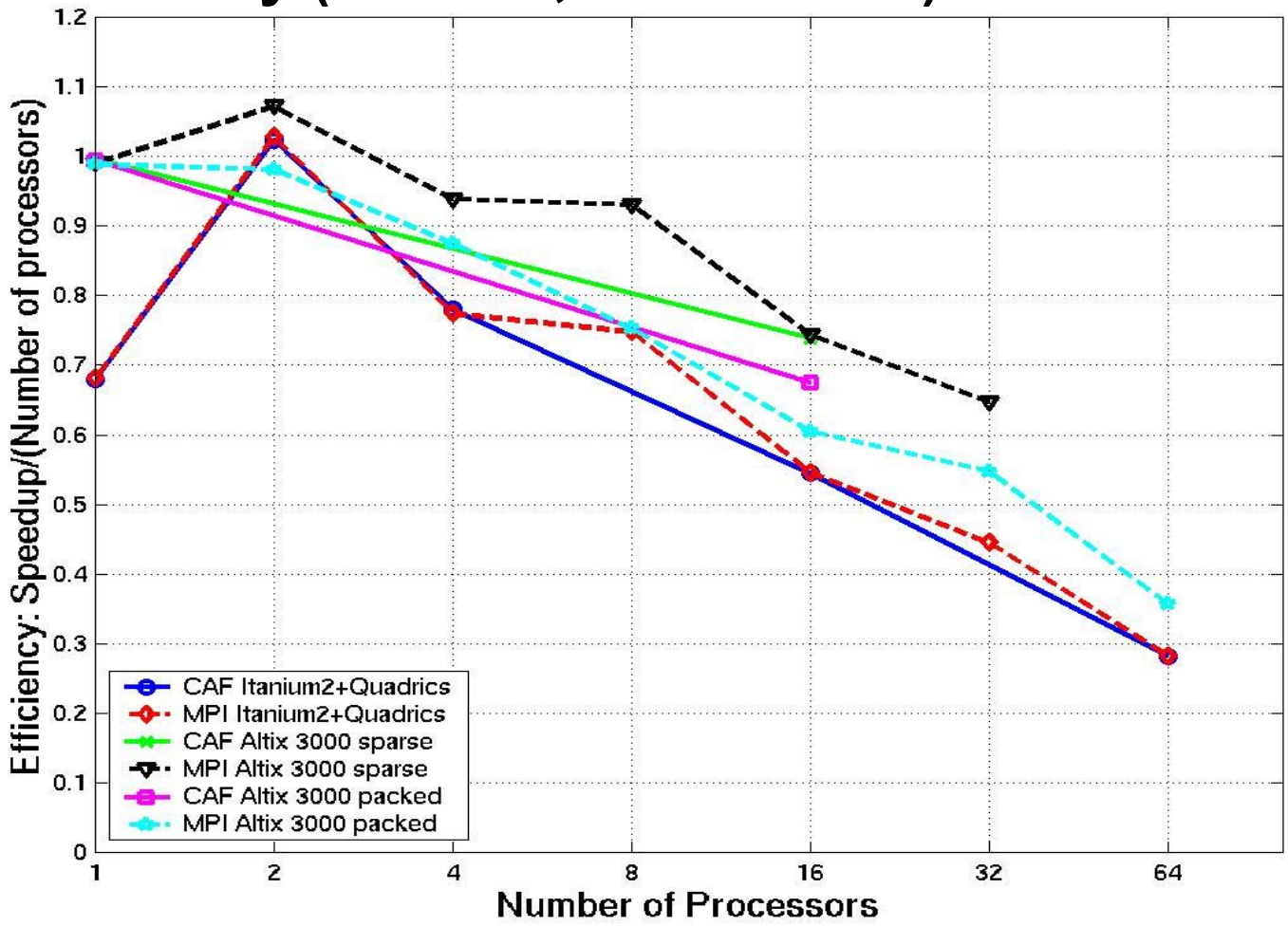
CAF: Оценка эффективности…

NPB MG Efficiency (Class C, size 512^3)



CAF: Оценка эффективности

NPB CG Efficiency (Class C, size 150000)



CAF 1.0 → CAF 2.0: Направления развития…

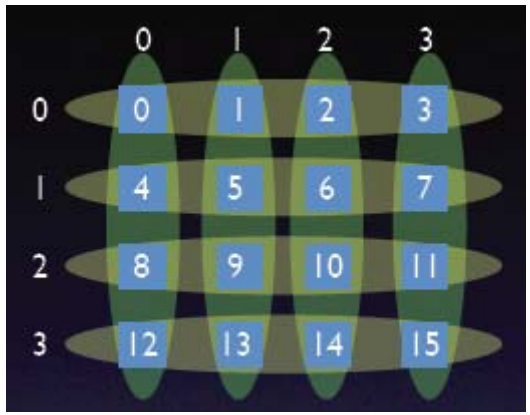
- ISO Fortran Committee решил включить элемента CAF в новый подготавливаемый стандарт языка Fortran
- Вместе с этим, активно ведутся работы по разработке следующей версии языка CAF (прежде всего усилиями исследователей Rice University). Основные цели при разработке CAF 2.0 состоят в следующем:
  - Расширить набор средств для разработки параллельных программ и библиотек параллельных методов,
  - Обеспечить большую эффективность параллельных вычислений,
  - Предоставить возможность эффективного выполнения CAF-программ на широком спектре параллельных вычислительных систем (от многоядерных процессоров до существенно многопроцессорных систем петафлопного уровня)

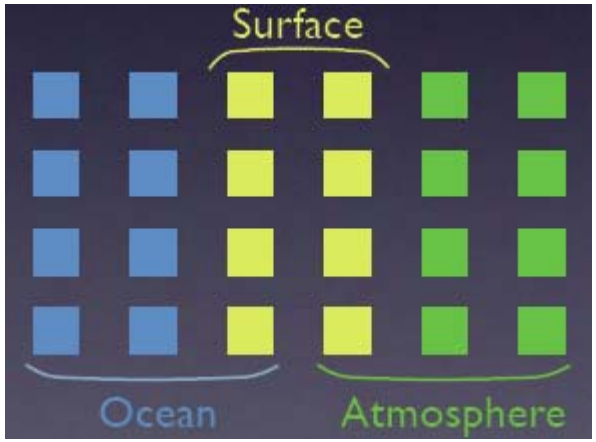
  и др.

CAF 1.0 → CAF 2.0: Направления развития…

Группы исполнителей…

- **Группа исполнителей** (team of images) –упорядоченный набор исполнителей
- Исполнитель может принадлежать нескольким группам
- Исполнитель в группе имеет свой индивидуальный индекс
- Группы могут быть использованы для создания распределенных массивов
- Группы являются основой для определения коллективных операций передачи данных





CAF 1.0 → CAF 2.0: Направления развития…

Группы исполнителей – создание через разделение существующей группы…

team_split (team, color, key, team_out)

- team – существующая группа исполнителей,
- color – индекс принадлежности к группе (исполнители с одним и тем же значением параметра color принадлежат одной и той же новой создаваемой группе),
- key – индекс исполнителя в новой группе,
- team_out – дескриптор новой создаваемой группы.

CAF 1.0 → CAF 2.0: Направления развития…

Группы исполнителей – создание через разделение существующей группы…

Пример:

- Пусть *p* исполнителей образуют сетку *q* x *q*
- Создадим отдельные группы для каждой строки и столбца

```
IMAGE_TEAM team
integer rank, row
rank = this_image(TEAM_WORLD)
row = rank/q
call team_split(TEAM_WORLD, row, rank, team)
```

CAF 1.0 → CAF 2.0: Направления развития…

Группы исполнителей – создание через разделение существующей группы

- Доступ к распределенному массиву через исполнитель группы

  x(i,j)[p@ocean] *! p есть ранг в группе ocean*

- Доступ с использованием правила по умолчанию “**with team**”

  with team atmosphere *! atmosphere как группа по умолчанию*

  x(:,0)[p] = y(:)[q@ocean] *! p – исполнитель из группы atmosphere, q – исполнитель из группы ocean*

  end with team

- Группы могут быть созданы также через объединение и пересечение существующих групп

CAF 1.0 → CAF 2.0: Направления развития…

Топологии исполнителей…

Обеспечение соответствия структуры взаимодействия исполнителей со структурой информационных связей

- **Создание топологии**
  - **topology_cartesian(/e1,e2,.../)** – декартова топология (решетка)
  - **topology_graph(n,e)** – топология общего графа
- **Изменение структуры топологии**
  - **graph_neighbor_add(g,e,n,nv)**
  - **graph_neighbor_delete(g,e,n,nv)**
- **Связывание группы исполнителей с топологией**
  - **topology_bind(team,topology)**

CAF 1.0 → CAF 2.0: Направления развития…

Топологии исполнителей…

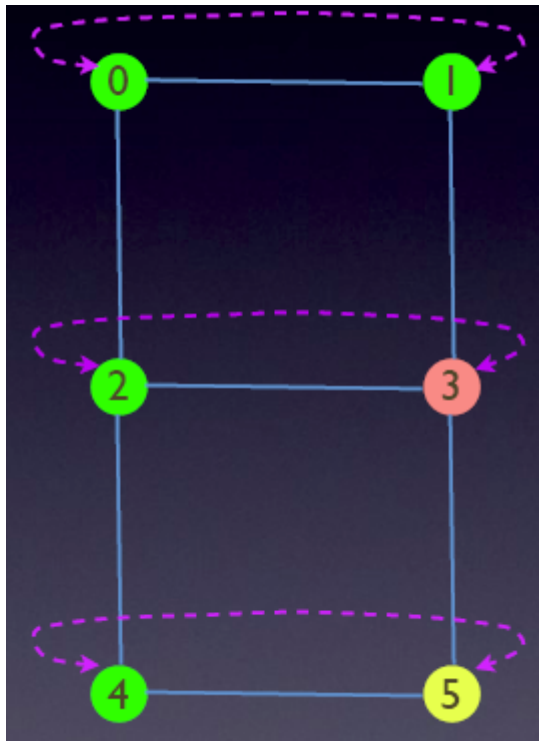
- **Доступ к распределенным массивам** с использованием топологий
  - **Декартова топология**
    - array(:) [**(i1, i2, ..., in)@ocean**] ! доступ из группы **ocean**
    - array(:) [**i1, i2, ..., in**] ! доступ из группы по умолчанию
  - **Общий граф**: доступ к **k**-му потомку исполнителя **i** для класса ребер **e**
    - array(:) [**(e,i,k)@ ocean**] ! доступ из группы **ocean**
    - array(:) [**e,i,k**] ! доступ из группы по умолчанию

CAF 1.0 → CAF 2.0: Направления развития…

Топологии исполнителей…

- **Пример: Декартова топология**

```
Integer, Allocatable :: X(:)[*], Y(100)[*]
Team :: Ocean, SeaSurface
! create a cartesian topology 2 (cyclic) by 3
Cart = Topology_cartesian( /-2, 3/ )
! bind teams Ocean and SeaSurface to Cart
Topology_bind( Ocean, Cart )
Topology_bind( SeaSurface, Cart )
! Ocean is the default team in this scope
With Team Ocean
  Allocate( X(100) )
  ! Y on node 3 gets X on node 5
  Y(:) [1, 1] = X(:)[ (-1, 2)@SeaSurface ]
End With Team
```

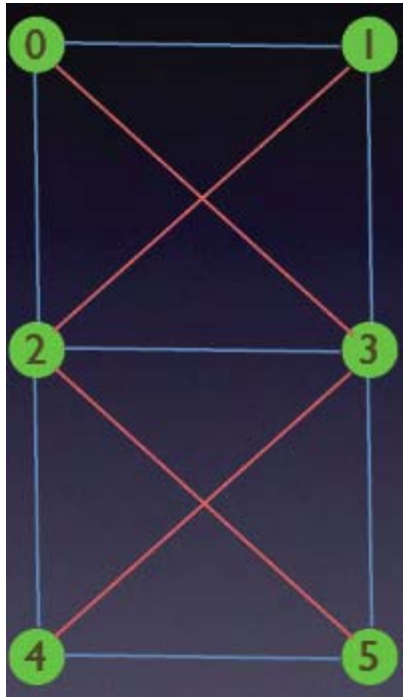


CAF 1.0 → CAF 2.0: Направления развития…

Топологии исполнителей

- **Пример: Топология в виде общего графа**

```
graph = topology_graph( 6, 2 )
integer :: red, blue, myrank
myrank = team_rank(team_world)
read *, blue_neighbors, red_neighbors
! blue edges
graph_neighbor_add(graph,blue,myrank,blue_neighbors)
! red edges
graph_neighbor_add(graph,red,myrank,red_neighbors)
! bind team with the topology
call topology_bind( ocean, graph )
allocate( x(100)@ocean )
! y receives x(20:80) from image 4
y(:) = x(20:80) [ (myrank, blue, 3)@ocean ]
```



CAF 1.0 → CAF 2.0: Направления развития

Коллективные операции

- Набор коллективных операций включает:
  - **co_bcast** - рассылка,
  - **co_gather** - сбор,
  - **co_allgather** – сбор и рассылка,
  - **co_permute** - перестановка,
  - **co_reduce** - редукция,
  - **co_allreduce** – редукция и рассылка,
  - **co_scan** – обобщенная редукция,
  - **co_scatter** – обобщенная рассылка,
  - **co_segmented_scan** – обобщенная редукция с сегментацией,
  - **co_shift** – сдвиг.
- Возможность двухэтапного исполнения для перекрытия вычислений и операций передачи данных

CAF: Заключение…

- Язык параллельного программирования CAF образован как незначительное расширение Fortran, достаточное для разработки эффективных параллельных программ
- Основу языка CAF составляют:
  - Понятие ***image*** (исполнитель) как абстракцию вычислительного элемента используемой компьютерной системы,
  - Понятие ***co-array*** (распределенного массива), компоненты которого распределены по исполнителям; доступ к распределенным компонентам осуществляется по правилам работы с обычными массивами
- Параллельная CAF-программа порождается копированием одного и того же программного кода (модель SPMD)

CAF: Заключение…

- ISO Fortran Committee планирует включить элемента CAF в новый подготавливаемый стандарт языка Fortran
- Вычислительные эксперименты показывают достаточную эффективность CAF-программ
- В предложениях по развитию CAF явно прослеживается влияние стандарта MPI. Среди основных предложений:
  - введение групп исполнителей,
  - возможность определения топологий,
  - наличие коллективных операций передачи данных

CAF: Литература

1. http://www.co-array.org
2. http://caf.rice.edu
3. Numrich, R. W. and Reid, J. K. (1998). Co-Array Fortran for parallel programming. ACM Fortran Forum (1998), 17, 2 (Special Report) and Rutherford Appleton Laboratory report RAL-TR-1998-060 available as ftp://ftp.numerical.rl.ac.uk/pub/reports/nrRAL98060.pdf
4. Numrich, R. W. and Reid, J. K. (2005). Co-arrays in the next Fortran Standard. ACM Fortran Forum (2005), 24, 2, 2-24 and
   WG5 paper
   ftp://ftp.nag.co.uk/sc22wg5/N1601-N1650/N1642.pdf
5. Numrich, R. W. and Reid, J. K. (2007). Co-arrays in the next Fortran Standard. Scientific Programming (2006), 14, 1-18.

Контакты:

Нижегородский университет

Факультет вычислительной математики и кибернетики

Гергель Виктор Павлович

gergel@unn.ru

СПАСИБО ЗА ВНИМАНИЕ

Вопросы ?