Microsoft Research

Introduction to Parallel Programming

Section 5.
Parallel programming language Co-Array Fortran

Victor Gergel, Professor, D.Sc.
Lobachevsky State University of Nizhni Novgorod (UNN)

Contents

- Parallel programming language Co-Array Fortran (CAF)
  - Approaches to parallel programs development
  - Parallel programming model CAF
  - *Images* (program images, executors)
  - *Co-arrays* (distributed arrays)
  - Synchronization of parallel computations
  - Examples
  - CAF programs performance measurement
  - Evolution: from CAF 1.0 to CAF 2.0

Approaches to parallel programs development

- **Usage of libraries** for existing programming languages – MPI for C and Fortran for distributed memory systems
- **Usage of “above-language” means (directives, comments)** - OpenMP for C and Fortran for shared memory systems
- **Extensions of existing programming languages** – e.g. UPC, **CAF**
- **Creation of new – parallel – programming languages** – e.g. **Chapel**, X10, Fortress,…

CAF: Introduction…

- Arrangement of computations according to PGAS model (**partitioned global address space**)
  - Two-level memory model for locality management (local/remote memory),
  - Management of data placement in local memory,
  - Control over data transfer from remote memory.
- CAF (Co-Array Fortran) was proposed by Numrich and Reid (Minnesota University, USA) in the mid 90s. The original name of the language – F^{--}.
- Language development supported by Cray company. One of the most active CAF development centers – Rice University.

Approach fundamentals – *minimal (but productive) extension of the Fortran language to make it an efficient parallel programming language*

CAF: Introduction

- Sample program

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

    ! send the data
    do i = 2, num_images()
      name[i] = name
    end do
  end if

  sync all ! wait for all images

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

CAF: Programming model…

- **Basis – Single-Program-Multiple-Data (SPMD) model:**
  - A single CAF-program is developed and further copied required number of times. CAF-program copies may run in parallel,
  - Each program copy has its own local data,
  - Data which has to be accessed from different copies of a CAF-program must be declared specifically *(co-arrays)*. Data transfer between program copies is done only via explicit syntax.

CAF: Programming model

- **A copy of a CAF-program is called *image* in CAF terminology:**
  - An image can be considered an abstraction of a computational node,
  - The number of created images can be determined using the **`num_images`**`()` function,
  - The number of images usually matches the number of available processors. In general case the two numbers can be different,
  - Each image is described by its unique *index* (**`this_image`**`()` function).

Indexes can be used to separate computations in images (in analogy with MPI programming)

CAF: Co-arrays…

- **To work with data shared by images CAF introduces *co-arrays* model.**
- **Syntax**

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

- **Semantics**

  Copying defined data object (array) to all images of CAF-program:

  – Number of co-array copy (image index) is specified using square brackets (index of a regular array is specified via round brackets). I.e. “()” brackets provide access to local memory and “[]” brackets – to remote memory,

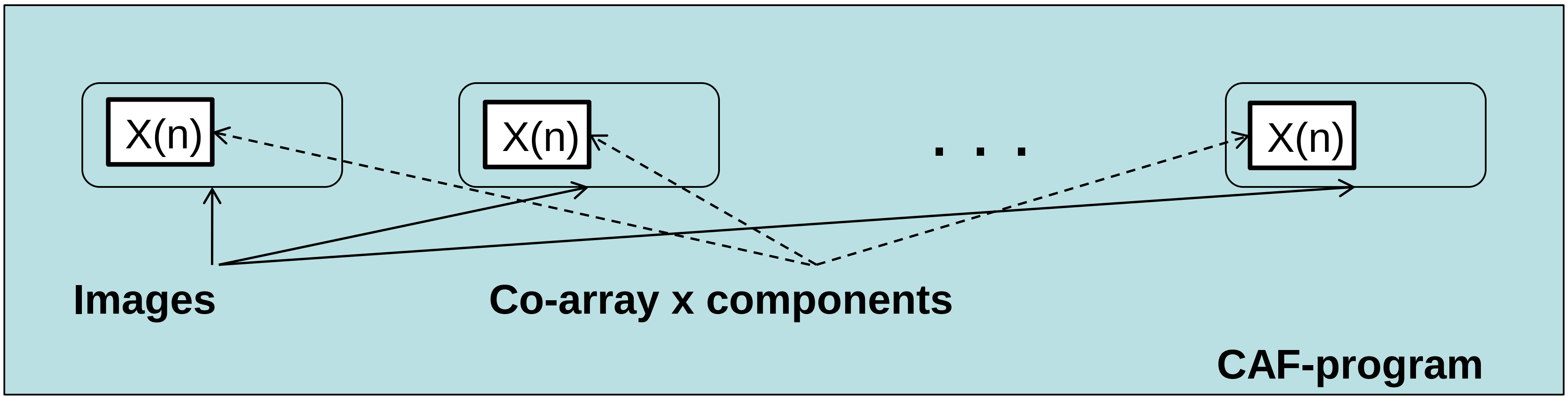
  – Number of copy local for the image can be omitted.

- **Example**

  – real :: x(n)[*] ! Creation of array **x** copies on all images of a program

CAF: Co-arrays…

- **Example**
  - real :: x(n)[*] ! Creation of array **x** copies on all images of a program



- **Operations on co-array components are performed according to ordinary Fortran rules**
  - **x(1)[p] = x(n)[q]** ! Read the element n of image *q* and
    ! write to an element 1 of image p

CAF: Co-arrays…

- **Co-array declaration can be done in full accordance with Fortran rules:**

```
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: Co-arrays…

❑ Example

```
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

- ❑ Maximum index value along the second dimension is determined as

  `= num_image()/4`

- ❑ The first index can be interpreted as a number of a core in a processor

CAF: Co-arrays…

❑ Example

```
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: Co-arrays…

❑ Example

```
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: Co-arrays…

- **Example**

```
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	-

- Images with indexes /-3,3/ and /-2,3/ are not defined

CAF: Co-arrays…

- **Example**

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

  - Images arrangement in a form of a three-dimensional grid
  - Maximum index value along the third dimension is defined as

```
= num_image()/(p*q)
```

CAF: Co-arrays

- **Operations on co-array components are also performed in full accordance with Fortran rules:**

```
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]
```

- Omitted index of co-array component indicates access to component which is local for the image

CAF: Syncronization…

Barrier synchronization…

- **Syntax**

```
sync all
```

- **Semantics**

Barrier synchronization – each image at the barrier is suspended until all images are at the barrier. I.e. segments which execute on an image before the **sync all** statement precede segments which execute after the **sync all** statement on any another image.

CAF: Synchronization…

Barrier synchronization

- **Example –** reading in image 1 and broadcasting to all others

```
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: Synchronization…

Synchronization of image groups…

- **Syntax**

```
sync images ( <image-set> )
```

- **Semantics**
  - Synchronization of images, specified as function argument; <image-set> parameter is an array of images' indexes for which the synchronization must be performed ("*" symbol can be used to specify all images).
  - **sync images (*)** is not equivalent to **sync all** – for **sync images** it is not required to use the same parameter value for all images which are synchronized (see example)

Synchronization of image groups

- **Example –** Synchronization of the image 1 with all other images (but not each with each)

```
if (this_image() == 1) then
  ! Setting data for all images
  sync images(*)
else
  sync images(1)
  ! Working with data
end if
```

CAF: Synchronization …

Synchronization of critical sections

- Syntax

```
critical
   ! Code executed by only one image
   ! at each moment of time
end critical
```

CAF: Synchronization

Memory synchronization

- **Syntax**

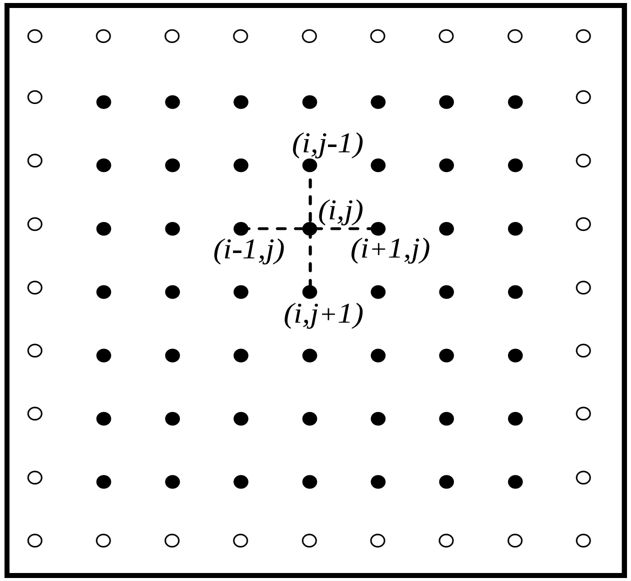
```
sync memory
```

- **Semantics**

  Synchronization of temporary and main memory
  (saving data which reside in temporary memory)

CAF: Example 1 – Finite Difference Method…

- The Dirichlet problem with periodic boundary conditions
- Finite Difference Method – 5-point kernel
- Structure of data – u(1:nrow)[1:ncol]



Gauss-Seidel method

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

CAF: Example 1 – Finite Difference Method…

```
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: Example 1 – Finite Difference Method

- **Explanation of the program:**
  - (1) – taking into account the periodic boundary conditions
  - (2) – adding values of horizontally adjacent cells
  - (3) – synchronization on readiness of previous computations
  - (4) – adding values of vertically adjacent cells

CAF: Example 2 – Search for Maximum in a Distributed Array…

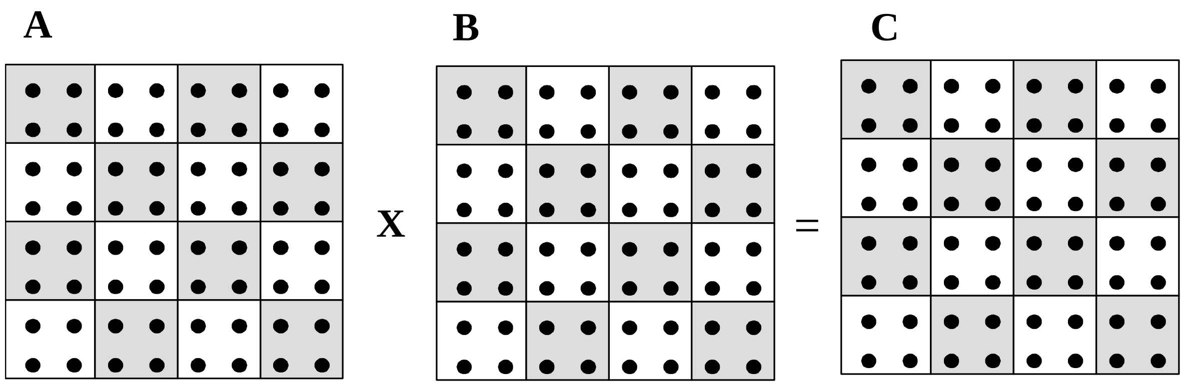
```
subroutine greatest(a,great) ! Find maximum value of a(:)[*]
  real, intent(in)  :: a(:)[*]
  real, intent(out) :: great[*]
  real :: work(num_images()) ! Local work array
  great = maxval(a(:))                                        (1)
  sync all ! Wait for all other images to reach here
  if(this_image(great)==1)then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

CAF: Example 2 – Search for Maximum in a Distributed array…

- **Explanation of the program:**
  - (1) – searching for local maximum by every image
  - (2) – getting all local maximums on the image 1
  - (3) – finding the global maximum and its broadcasting for all images

CAF: Example 3 – Matrix Multiplication…

- **Data distribution** – Block partitioning



- **Base subtask** – procedure of calculation of all elements for one of the matrix C blocks

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

CAF: Example 3 – Matrix Multiplication…

Version 1

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

! Calculating the matrix C block (myP,myQ)
! Block size n×n
! Images – p×p grid
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: Example 3 – Matrix Multiplication

Version 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: Performance Measurement…

Performance measurement with NASA Parallel Benchmark (NPB)…

- Developed in the early 90s
- Was developed as a universal tool for supercomputers performance measurement
- Includes kernels of hydro- and aerodynamic modeling problems
- Officially is just a set of rules and recommendations
- Reference implementation is available on the NASA server

CAF: Performance Measurement …

Performance measurement with NASA Parallel Benchmark (NPB)…

- EP — Embarrassing Parallel. Numerical integration with Monte-Carlo method.
- MG — simple 3D MultiGrid benchmark. Approximate the solution to a three-dimensional discrete Poisson equation («3D grid") on a NxNxN grid with periodic boundary conditions.
- CG — solving an unstructured sparse linear system by the Conjugate Gradient method. Estimate the smallest eigenvalue of a large sparse symmetric positive-definite matrix using the inverse iteration with the conjugate gradient method.
- FT —3-D Fast-Fourier Transform partial differential equation benchmark. Solve a three-dimensional partial differential equation (PDE) using the fast Fourier transform (FFT).
- IS — Parallel Sort of small Integers. Parallel sort of N integer numbers.
- LU — LU Solver. Solve a synthetic system of nonlinear PDEs using symmetric successive over-relaxation (SSOR) solver kernel.
- SP — Scalar Pentadiagonal. Solve a synthetic system of nonlinear PDEs using scalar pentadiagonal algorithm .
- BT — Block Tridiagonal. Solve a synthetic system of nonlinear PDEs using block tridiagonal algorithm.

Test	Class A	Class B	Class C	Class D	Class 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: Performance Measurement…

Performance measurement with NASA Parallel Benchmark (NPB)…

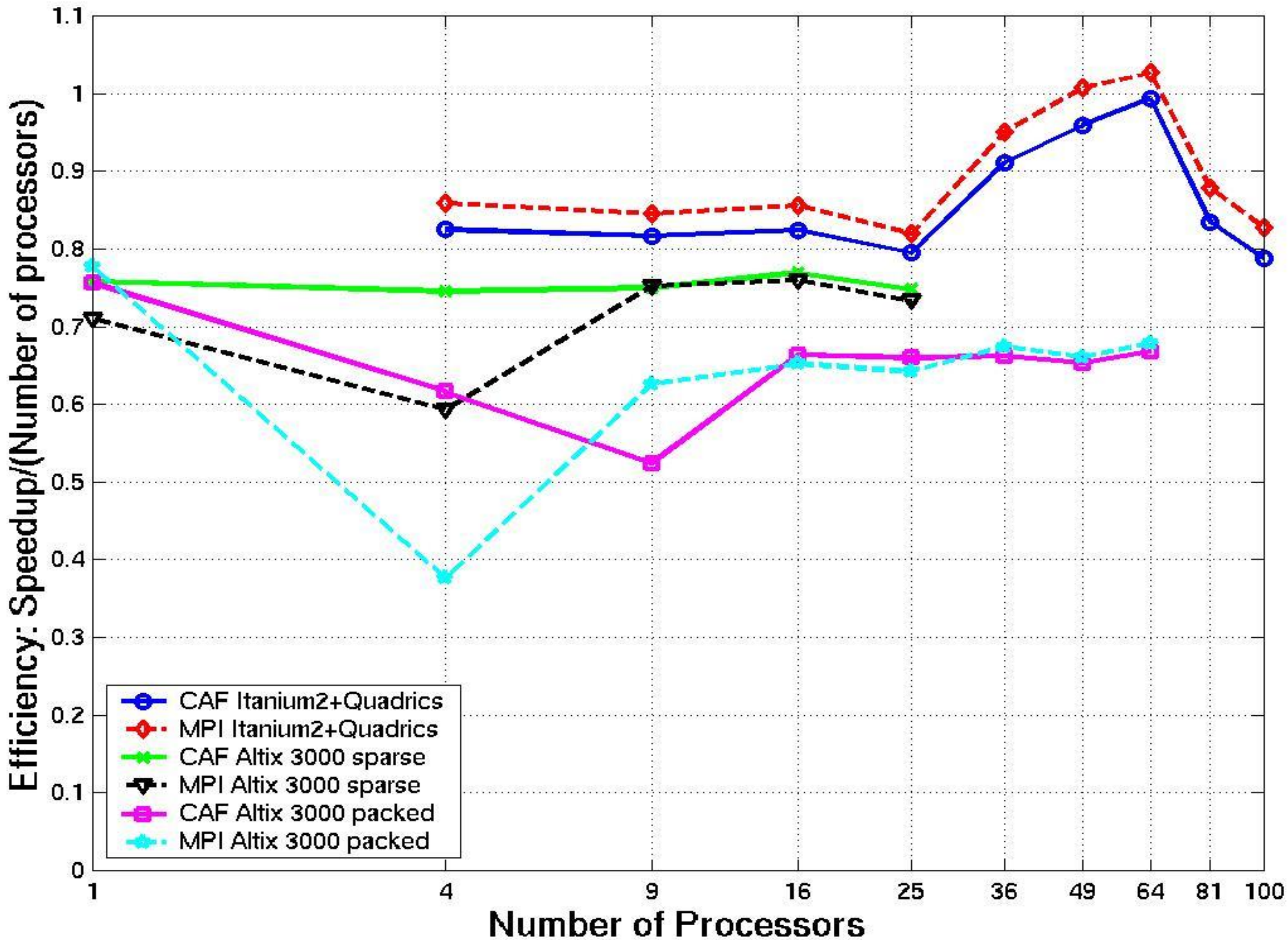
- NAS versions:
  - NPB2.3b2 : MPI implementation
  - NPB2.3-serial : Sequential code on the base of MPI version
- CAF version
  - NPB2.3-CAF: CAF implementation on the base of MPI version
- Hardware platforms
  - SGI Altix 3000 (Itanium2 1.5GHz)
  - Itanium2+Quadrics QSNet II (Elan4, Itanium2 1.5GHz)

Results of numerical experiments are taken from:

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.

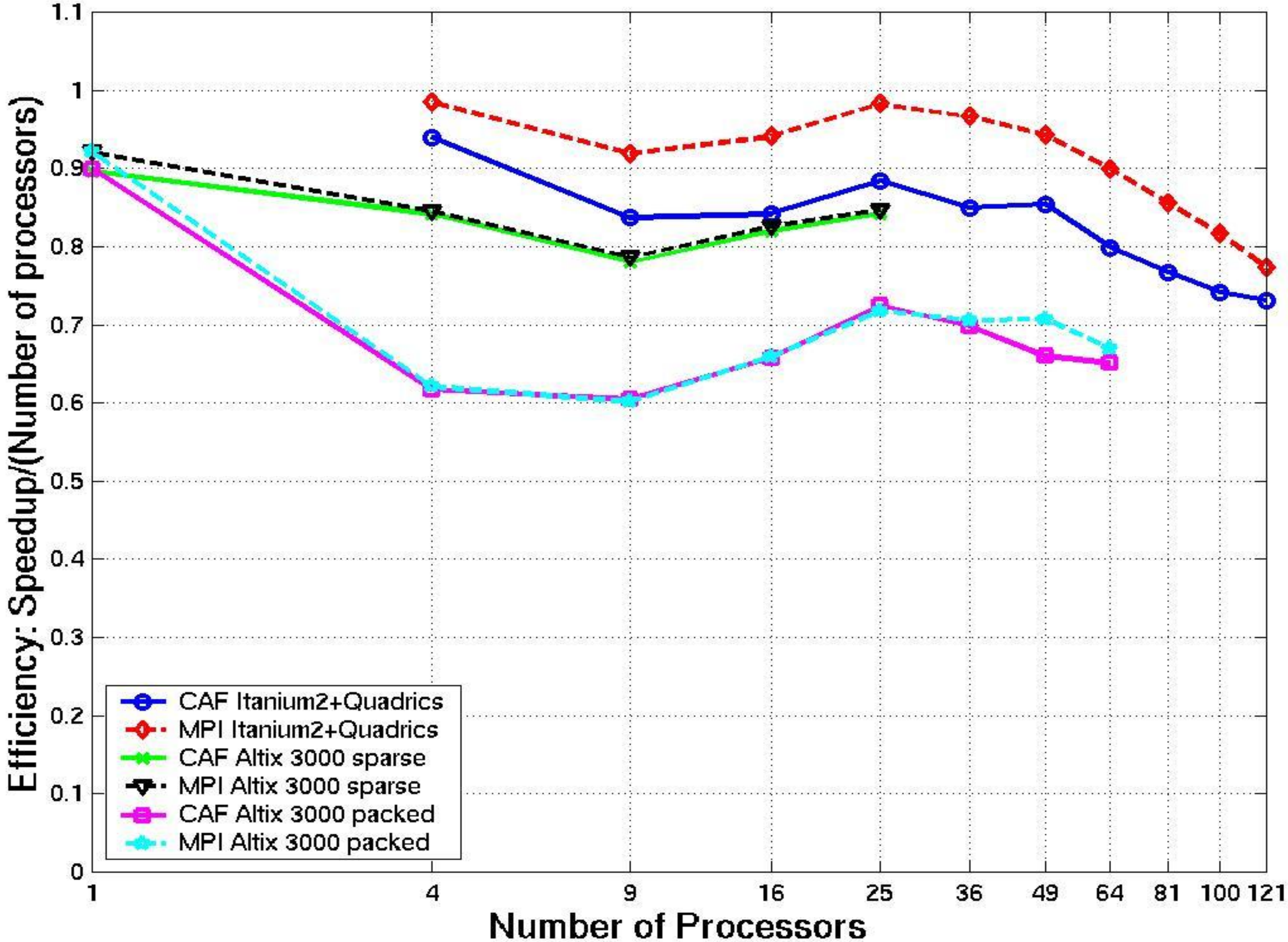
CAF: Performance Measurement…

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



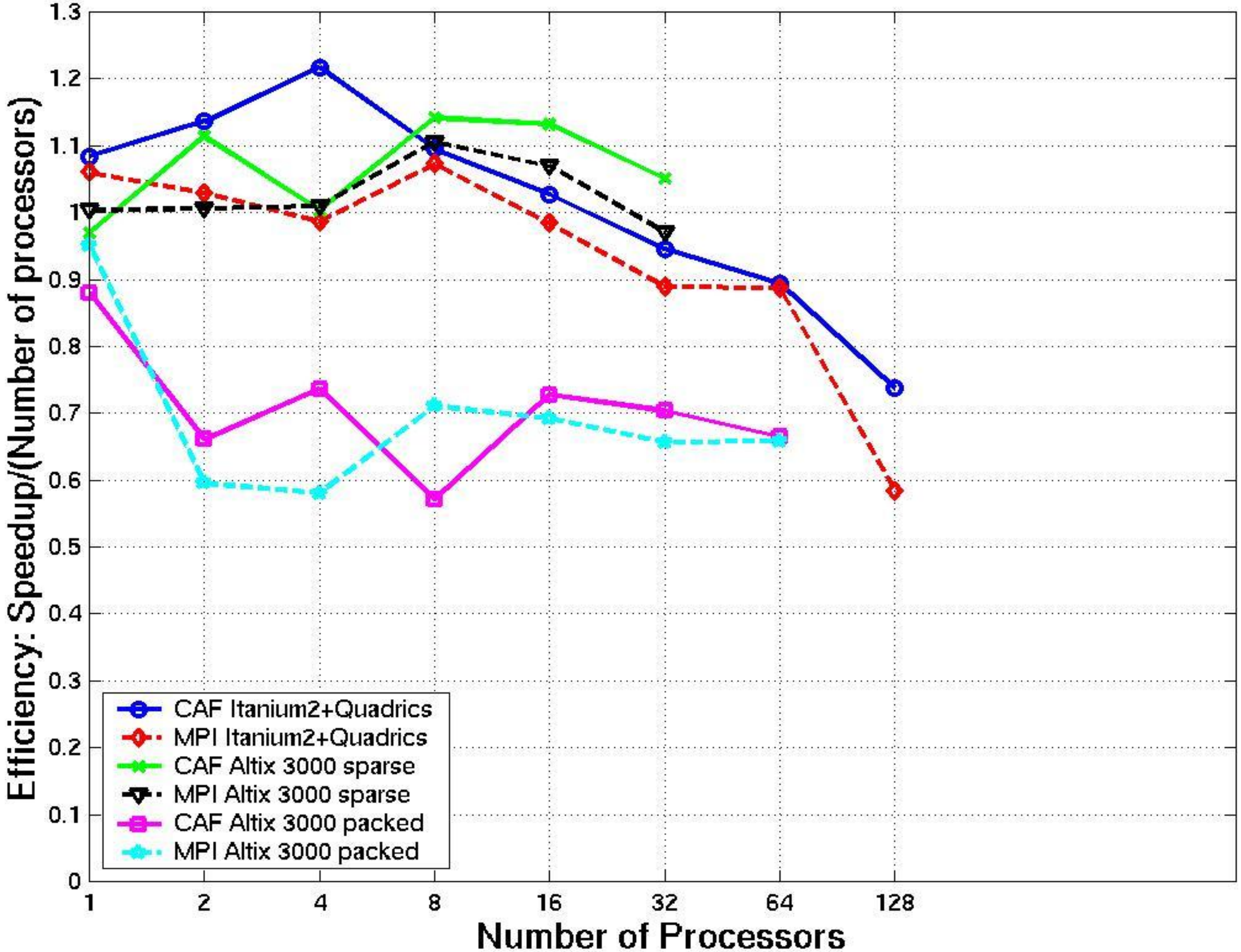
CAF: Performance Measurement…

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



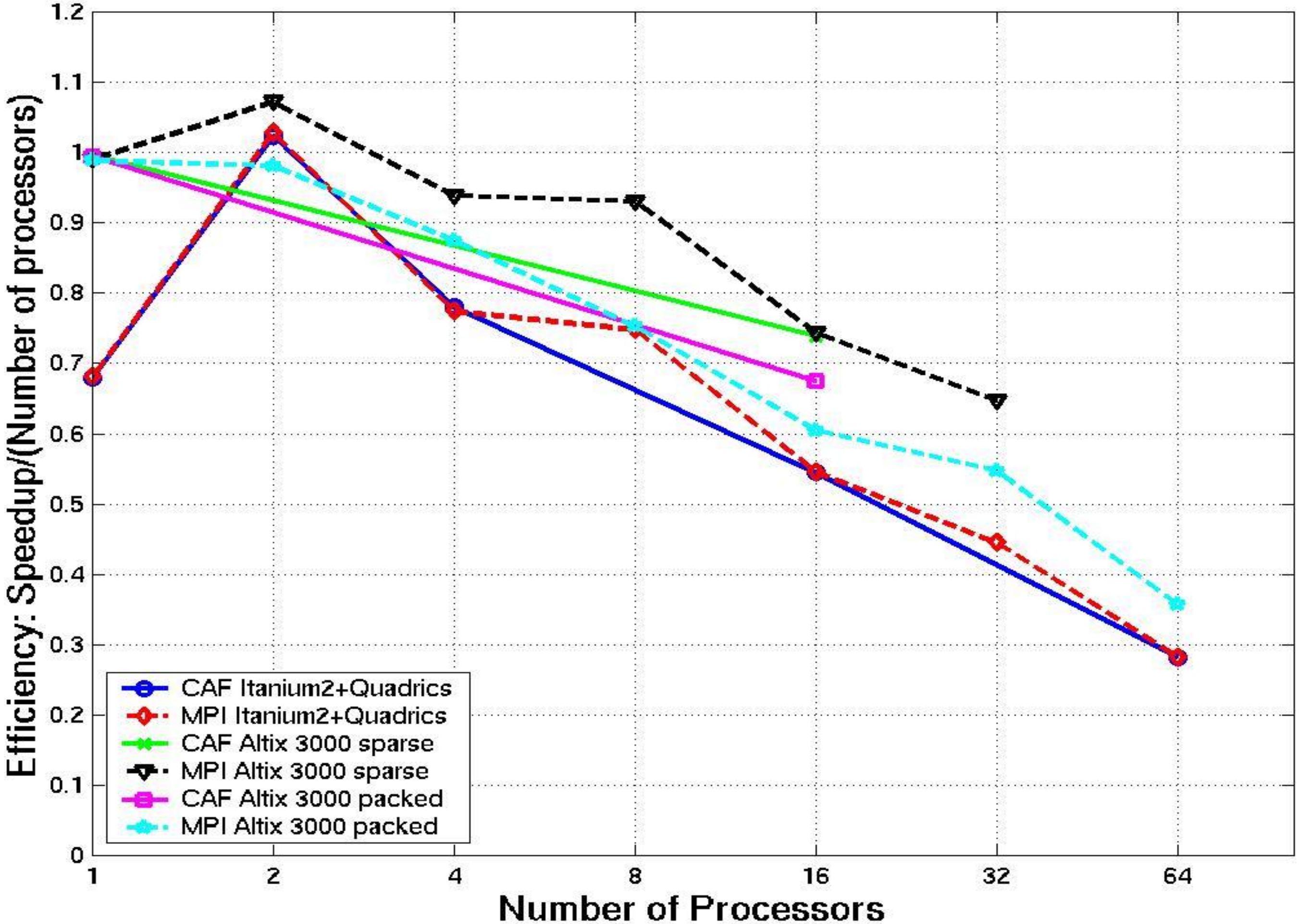
CAF: Performance Measurement…

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



CAF: Performance Measurement

NPB CG Efficiency (Class C, size 150000)



CAF 1.0 → CAF 2.0: Directions of Development…

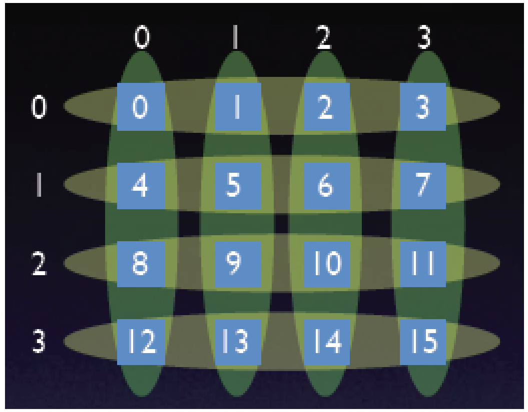
- ISO Fortran Committee decided to include CAF elements in the new standard of the Fortran language
- At the same time a new version of CAF language is actively developed (mostly by Rice University researchers). Major goals of the CAF 2.0 are the following:
  - Expand the set of features for development of parallel programs and libraries of parallel methods
  - Provide better efficiency of parallel computations
  - Provide the possibility of efficient execution of CAF-programs on a wide spectrum of parallel computing systems (from multicore processors to massively multiprocessor systems of the PetaFLOP level)

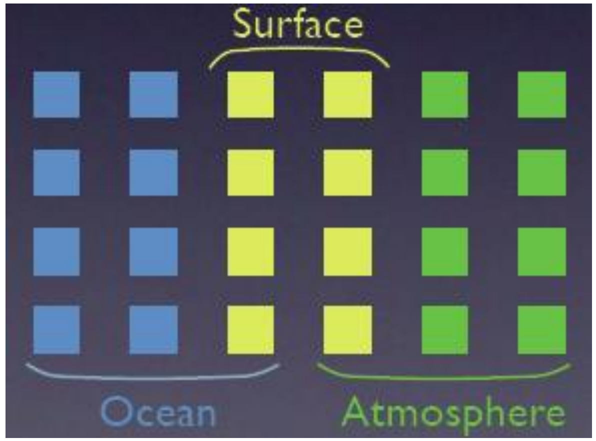
  etc.

CAF 1.0 → CAF 2.0: Directions of Development…

Teams of images…

- Team of images – ordered set of images
- Image can belong to several teams
- Image has its individual index in a team
- Teams can be used for creation of distributed arrays
- Teams are the base for defining collective data transfer operations





CAF 1.0 $\rightarrow$ CAF 2.0: Directions of Development…

Teams of images – creation via splitting an existing team…

team_split (team, color, key, team_out)

- team – existing team of images,
- color – index showing belonging to a team (images with the same color value belong to the same newly created team),
- key – index of an image in the new team,
- team_out – descriptor of the new team.

CAF 1.0 → CAF 2.0: Directions of Development…

Teams of images – creation via splitting an existing team…

Example:

- Assume *p* images form a *q* x *q* grid
- We create separate teams for every row and column

```
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: Directions of Development…

Teams of images – creation via splitting an existing team…

- Accessing co-arrays using a team image

  x(i,j)[p@ocean] ! *p* is a rank in the *ocean* team

- Accessing using the "**with team**" default rule

  with team atmosphere ! *atmosphere* as a default team

  x(:,0)[p] = y(:)[q@ocean] ! *p* – image from the *atmosphere* team,
  ! *q* – image from the *ocean* team

  end with team

- Teams can be also created via union and intersection of existing teams.

CAF 1.0 → CAF 2.0: Directions of Development…

Topologies of images…

Provides consistency of images' communications structure with informational network structure

- ❑ **Topology creation**

  - topology_cartesian(/e1,e2,.../) – Cartesian topology (grid)

  - **topology_graph(n,e)** – General graph topology

- ❑ **Changing the structure of topology**

  - **graph_neighbor_add(g,e,n,nv)**

  - **graph_neighbor_delete(g,e,n,nv)**

- ❑ **Binding a team of images with a topology**

  - **topology_bind(team,topology)**

CAF 1.0 → CAF 2.0: Directions of Development…

Topologies of images…

- Accessing co-arrays using a topology
  - **Cartesian topology**
    - array(:) [**(i1, i2, ..., in)@ocean**] ! Access from the *ocean* team
    - array(:) [**i1, i2, ..., in**] ! Access from the default team
  - **General graph topology**: Accessing the ***k*-th** neighbor of an image ***i*** in the edge class **e**
    - array(:) [**(e,i,k)@ ocean**] ! Access from the *ocean* team
    - array(:) [**e,i,k**] ! Access from the default team

CAF 1.0 → CAF 2.0: Directions of Development…

Topologies of images…

❑ Example: Cartesian topology

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

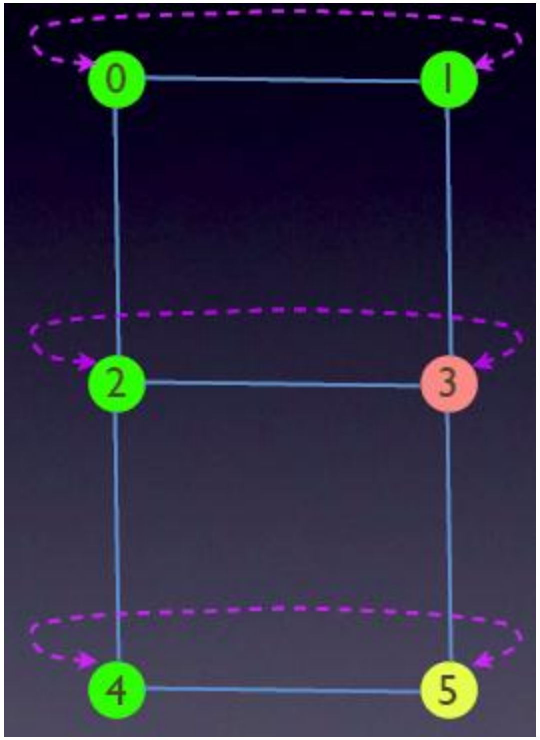
With Team Ocean

  Allocate(X(100))

  ! Y on node 3 gets X on node 5

  Y(:) [1, 1] = X(:)[(-1, 2)@SeaSurface]

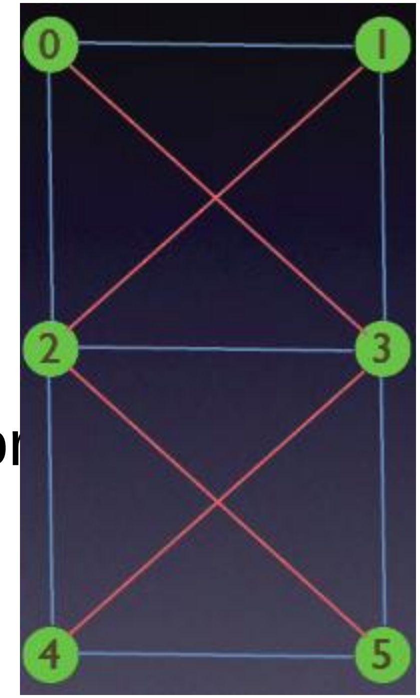
End With Team



CAF 1.0 → CAF 2.0: Directions of Development…

Topologies of images

❑ **Example: General graph topology**




```
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,blu46e,myrank,blue_neighbor
! 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: Directions of Development

Collective operations

- Set of collective operations includes:
  - **`co_bcast`** - broadcasting,
  - **`co_gather`** - gathering,
  - **`co_allgather`** – gathering and broadcasting,
  - **`co_permute`** - permutation,
  - **`co_reduce`** - reduction,
  - **`co_allreduce`** – reduction and broadcasting,
  - **`co_scan`** – generalized reduction,
  - **`co_scatter`** – generalized gathering,
  - **`co_segmented_scan`** – generalized reduction with segmentation,
  - **`co_shift`** – shift.
- The 2-stage execution possibility for interleaving calculations and data transfer operations.

CAF: Conclusions…

- CAF parallel programming language is formed as a small Fortran extension sufficient for development of efficient parallel programs.
- CAF is based on the following concepts:
  - ***image*** as an abstraction of a computing element of the computer system in use,
  - ***co-array*** (distributed array) with components distributed between images; distributed components are accessed according to the rules of regular array operations.
- Parallel CAF-program is executed by copying the same source code (SPMD model).

CAF: Conclusions…

- ISO Fortran Committee is going to include elements of CAF in the new prepared standard of the Fortran language
- Computational experiments show sufficient efficiency of CAF-programs
- Influence of MPI standard can be traced in CAF improvement proposals. Major proposals:
  - introducing teams of images,
  - the possibility to declare a topology,
  - presence of collective data transfer operations

CAF: References

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.

Contacts:

Nizhny Novgorod State University

Department of Computational Mathematics and Cybernetics

Victor P. Gergel

gergel@unn.ru

Thank you for attention!

Any questions?