

**Автоматизация разработки  
параллельных программ для  
современных  
высокопроизводительных ЭВМ**

**В.А. Крюков**

**Факультет ВМК МГУ,  
Институт прикладной математики им. М.  
В. Келдыша РАН  
[krukov@keldysh.ru](mailto:krukov@keldysh.ru)**

**25.08.2012**

# План изложения

- Модели и языки программирования с явным параллелизмом (абстрактная и целевая машина, уровень, компилятор/библиотека, новый язык/расширение существующего, новые конструкции/директивы компилятору)
- Языки программирования с неявным параллелизмом + автоматическое распараллеливание (**Fortran**, C, C++)
- Автоматизация преобразования имеющихся последовательных программ в эффективные параллельные программы на языках с явным или неявным параллелизмом
- Автоматизация отладки

# Модели и языки программирования с явным параллелизмом

- абстрактная параллельная машина и целевая ЭВМ:  
**многоядерный кластер с ускорителями/**  
многоядерный кластер/ одноядерный кластер/  
узел кластера/ много ускорителей/ один ускоритель
- **высокий** или **низкий** уровень, **глобальный** или **фрагментарный** взгляд
- **компилятор** или **библиотека**
- **новый язык** или **расширение существующего**
- **новые конструкции** или **директивы компилятору**

# Модели и языки с явным параллелизмом

<i>Chapel, X10</i>				
<i>DVMH</i>				
<i>DVM/OpenMP</i>				
HPF, <b>DVM</b> , CAF-UPC	<b>OpenMP</b>		PGI_APM, OpenACC	
<b>MPI</b>	Shmem	Pthreads	<b>CUDA</b> -Op enCL	(RCCE-M CAPI) ?

# Алгоритм Якоби на языке Fortran

```
PROGRAM JACOB_SEQ
  PARAMETER (L=8, ITMAX=20)
  REAL A(L,L), B(L,L)

  PRINT *, '***** TEST_JACOBI *****'

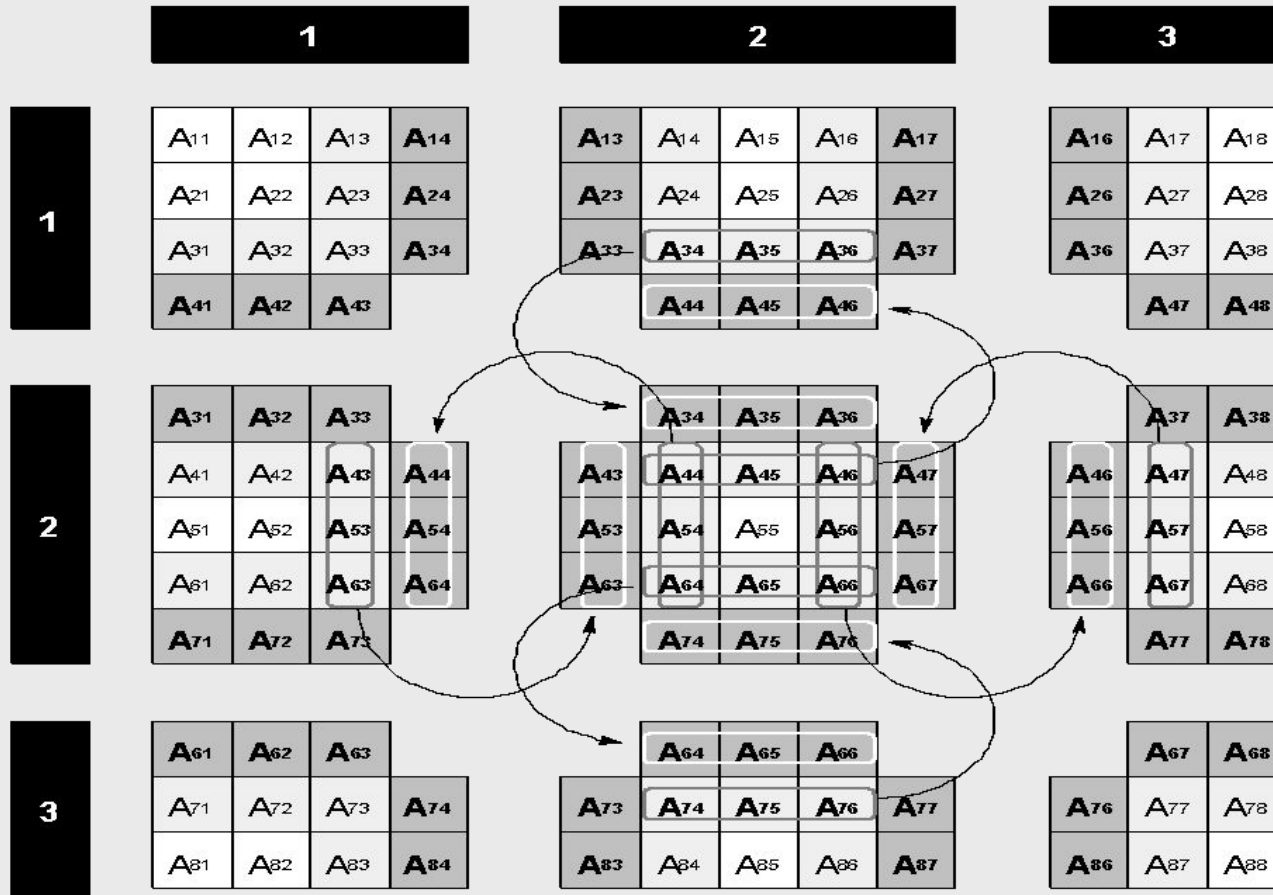
  DO IT = 1, ITMAX

    DO J = 2, L-1
      DO I = 2, L-1
        A(I, J) = B(I, J)
      ENDDO
    ENDDO

    DO J = 2, L-1
      DO I = 2, L-1
        B(I, J) = (A(I-1, J) + A(I, J-1) + A(I+1, J) +
*                A(I, J+1)) / 4
      ENDDO
    ENDDO
  ENDDO
END
```

# Distribution of array A [8][8]

processor arrangement 3\*3



```
PROGRAM JACOB_MPI
```

```
include 'mpif.h'
```

```
integer me, nprocs
```

```
PARAMETER (L=4096, ITMAX=100, LC=2, LR=2)
```

```
REAL A(0:L/LR+1,0:L/LC+1), B(L/LR,L/LC)
```

```
C arrays A and B with block distribution
```

```
integer dim(2), coords(2)
```

```
logical isper(2)
```

```
integer status(MPI_STATUS_SIZE,4), req(8), newcomm
```

```
integer srow,lrow,nrow,scol,lcol,ncol
```

```
integer pup,pdown,pleft,pright
```

```
dim(1) = LR
```

```
dim(2) = LC
```

```
isper(1) = .false.
```

```
isper(2) = .false.
```

```
reor = .true.
```

```
call MPI_Init( ierr )
```

```
call MPI_Comm_rank( mpi_comm_world, me, ierr )
```

```
call MPI_Comm_size( mpi_comm_world, nprocs, ierr)
```

```
call MPI_Cart_create( mpi_comm_world, 2, dim, isper,  
* .true., newcomm, ierr)
```

```
call MPI_Cart_shift( newcomm, 0, 1, pup, pdown, ierr)
```

```
call MPI_Cart_shift( newcomm, 1, 1, pleft, pright, ierr)
```

```
call MPI_Comm_rank( newcomm, me, ierr)
```

```
call MPI_Cart_coords( newcomm, me, 2, coords, ierr)
```

```

C   rows of matrix I have to process
      srow = (coords(1) * L) / dim(1)
      lrow = (((coords(1) + 1) * L) / dim(1)) - 1
      nrow = lrow - srow + 1
C   colomns of matrix I have to process
      scol = (coords(2) * L) / dim(2)
      lcol = (((coords(2) + 1) * L) / dim(2)) - 1
      ncol = lcol - scol + 1
      call MPI_Type_vector(ncol,1,nrow+2,MPI_DOUBLE_PRECISION,
* vectype, ierr)

      call MPI_Type_commit(vectype, ierr)

      IF (me .eq. 0) PRINT *, '***** TEST_JACOBI *****'

      DO IT = 1, ITMAX
          DO J = 1, ncol
              DO I = 1, nrow
                  A(I, J) = B(I, J)
              ENDDO
          ENDDO
      ENDDO

C   Copying shadow elements of array A from
C   neighbouring processors before loop execution

```



```

    call MPI_Irecv(A(1,0),nrow,MPI_DOUBLE_PRECISION,
*      pleft, 1235, MPI_COMM_WORLD, req(1), ierr)
    call MPI_Isend(A(1,ncol),nrow,MPI_DOUBLE_PRECISION,
*      pright, 1235, MPI_COMM_WORLD,req(2), ierr)
    call MPI_Irecv(A(1,ncol+1),nrow,MPI_DOUBLE_PRECISION,
*      pright, 1236, MPI_COMM_WORLD, req(3), ierr)
    call MPI_Isend(A(1,1),nrow,MPI_DOUBLE_PRECISION,
*      pleft, 1236, MPI_COMM_WORLD,req(4), ierr)
    call MPI_Irecv(A(0,1),1,vectype,
*      pup, 1237, MPI_COMM_WORLD, req(5), ierr)
    call MPI_Isend(A(nrow,1),1,vectype,
*      pdown, 1237, MPI_COMM_WORLD,req(6), ierr)
    call MPI_Irecv(A(nrow+1,1),1,vectype,
*      pdown, 1238, MPI_COMM_WORLD, req(7), ierr)
    call MPI_Isend(A(1,1),1,vectype,
*      pup, 1238, MPI_COMM_WORLD,req(8), ierr)
    call MPI_Waitall(8,req,status, ierr)
      DO J = 2, ncol-1
        DO I = 2, nrow-1
          B(I, J) = (A( I-1, J ) + A( I, J-1 ) +
*            A( I+1, J) + A( I, J+1 )) / 4
        ENDDO
      ENDDO
    ENDDO
    call MPI_Finalize(ierr)
  END

```

PROGRAM JACOB\_OpenMP

PARAMETER (L=4096, ITMAX=100)

REAL A(L,L), B(L,L)

PRINT \*, '\*\*\*\*\* TEST\_JACOBI \*\*\*\*\*'

!\$OMP PARALLEL

DO IT = 1, ITMAX

!\$OMP DO

DO J = 2, L-1

DO I = 2, L-1

A(I,J) = B(I,J)

ENDDO

ENDDO

!\$OMP DO

DO J = 2, L-1

DO I = 2, L-1

B(I,J) = (A(I-1,J) + A(I,J-1) +

\* A(I+1,J) + A(I,J+1)) / 4

ENDDO

ENDDO

ENDDO

!\$OMP END PARALLEL

END

```
module jac_cuda
```

```
contains
```

```
attributes(global) subroutine arr_copy(a, b, k)
```

```
real, device, dimension(k, k) :: a, b
```

```
integer, value :: k
```

```
integer i, j
```

```
i = (blockIdx%x - 1) * blockDim%x + threadIdx%x
```

```
j = (blockIdx%y - 1) * blockDim%y + threadIdx%y
```

```
if (i.ne.1 .and. i.ne.k .and. j.ne.1 .and. j.ne.k) then
```

```
a(i, j) = b(i, j)
```

```
endif
```

```
end subroutine arr_copy
```

```
attributes(global) subroutine arr_renew(a, b, k)
```

```
real, device, dimension(k, k) :: a, b
```

```
integer, value :: k
```

```
integer i, j
```

```
    i = (blockIdx%x - 1) * blockDim%x + threadIdx%x
```

```
        j = (blockIdx%y - 1) * blockDim%y + threadIdx%y
```

```
if (i.ne.1 .and. i.ne.k .and. j.ne.1 .and. j.ne.k) then
```

```
b(i, j) = (a(i-1, j) + a(i+1, j) + a(i, j-1) + a(i, j+1))/4
```

```
endif
```

```
end subroutine arr_renew
```

```
end module jac_cuda
```

# program JACOB\_CUDA

```
use cudafor
use jac_cuda

parameter (k=4096, itmax = 100, block_dim = 16)
real, device, dimension(k, k) :: a, b
integer it
type(dim3) :: grid, block

print *, '***** test_jacobi *****'

grid = dim3(k / block_dim, k / block_dim, 1)
block = dim3(block_dim, block_dim, 1)

do it = 1, itmax
  call arr_copy<<<grid, block>>>(a, b, k)
  call arr_renew<<<grid, block>>>(a, b, k)
end do

end program jacob_CUDA
```

```
PROGRAM JACOB_PGI_APM
```

```
PARAMETER (L=4096, ITMAX=100)
```

```
REAL A(L,L), B(L,L)
```

```
PRINT *, '***** TEST_JACOBI *****'
```

```
!$acc data region copyout(B), local(A)
```

```
DO IT = 1, ITMAX
```

```
!$acc region
```

```
DO J = 2, L-1
```

```
DO I = 2, L-1
```

```
A(I,J) = B(I,J)
```

```
ENDDO
```

```
ENDDO
```

```
DO J = 2, L-1
```

```
DO I = 2, L-1
```

```
B(I,J) = (A(I-1,J) + A(I,J-1) +
```

```
* A(I+1,J) + A(I,J+1)) / 4
```

```
ENDDO
```

```
ENDDO
```

```
!$acc end region
```

```
ENDDO
```

```
!$acc end data region
```

```
END
```

# Автоматическое распараллеливание

Для распределенных систем проблематично в силу следующих причин:

- вычислительная работа должна распределяться между процессорами крупными порциями
- на системах с распределенной памятью необходимо произвести не только распределение вычислений, но и распределение данных, а также обеспечить на каждом процессоре доступ к удаленным данным - данным, расположенным на других процессорах
- распределение вычислений и данных должно быть произведено согласованно

# Модель параллелизма по данным

- отсутствует понятие процесса и, как следствие, явная передача сообщений или явная синхронизация
- данные последовательной программы распределяются программистом по процессорам параллельной машины
- последовательная программа преобразуется компилятором в параллельную программу
- вычисления распределяются по правилу собственных вычислений: каждый процессор выполняет только вычисления собственных данных

# HPF (High Performance Fortran)

**HPF** (1993 год) - расширение языка Фортран 90

**HPF-2** (1997 год) - расширение языка Фортран 95

Распределение данных производится в два этапа

- с помощью директивы **ALIGN** задается соответствие между взаимным расположением элементов нескольких массивов
- группа массивов с помощью директивы **DISTRIBUTE** отображается на решетку процессоров

Заданное распределение данных может быть изменено на этапе выполнения программы с помощью операторов **REALIGN** и **REDISTRIBUTE**



```

PROGRAM      JAC_HPF
PARAMETER    (L=8,  ITMAX=20)
REAL        A(L,L), B(L,L)

```

```

!HPF$ PROCESSORS P(3,3)

```

```

!HPF$ DISTRIBUTE      ( BLOCK,  BLOCK)      ::  A

```

```

!HPF$ ALIGN  B(I,J)  WITH  A(I,J)

```

```

C arrays A and B with block distribution

```

```

PRINT *, '***** TEST_JACOBI *****'

```

```

DO IT = 1, ITMAX

```

```

!HPF$ INDEPENDENT

```

```

DO J = 2, L-1

```

```

!HPF$ INDEPENDENT

```

```

DO I = 2, L-1

```

```

A(I, J) = B(I, J)

```

```

ENDDO

```

```

ENDDO

```

```

!HPF$ INDEPENDENT

```

```

DO J = 2, L-1

```

```

!HPF$ INDEPENDENT

```

```

DO I = 2, L-1

```

```

B(I, J) = (A(I-1, J) + A(I, J-1) +

```

```

* A(I+1, J) + A(I, J+1)) / 4

```

```

ENDDO

```

```

ENDDO

```

```

ENDDO

```

```

END

```

PROGRAM JACOB\_DVM

PARAMETER (L=4096, ITMAX=100)

REAL A(L,L), B(L,L)

CDVM\$ DISTRIBUTE (BLOCK, BLOCK) :: A

CDVM\$ ALIGN B(I,J) WITH A(I,J)

C arrays A and B with block distribution

PRINT \*, '\*\*\*\*\* TEST\_JACOBI \*\*\*\*\*'

DO IT = 1, ITMAX

CDVM\$ PARALLEL (J,I) ON A(I,J)

DO J = 2, L-1

DO I = 2, L-1

A(I,J) = B(I,J)

ENDDO

ENDDO

CDVM\$ PARALLEL (J,I) ON B(I,J), SHADOW\_RENEW (A)

C Copying shadow elements of array A from

C neighboring processors before loop execution

DO J = 2, L-1

DO I = 2, L-1

B(I,J) = (A(I-1,J) + A(I,J-1) +

\* A(I+1,J) + A(I,J+1)) / 4

ENDDO

ENDDO

ENDDO

END

# Новые языки параллельного программирования (PGAS)

PGAS – Partitioned Global Address Space (не DSM !)  
CAF (Co-Array Fortran), UPC (Unified Parallel C),  
Titanium (расширение Java)

2005 г.:

- Много нитей выполняют одну программу в стиле SPMD
- Уровень языков не намного выше MPI
- Нет аналога коммутаторов
- Нет средств совмещения обменов с вычислениями

# Coarray Fortran

```
PROGRAM Jac_CoArray
  PARAMETER (L=8, ITMAX=20, LR=2, LC=2)
  PARAMETER (NROW=L/LR, NCOL=L/LC)
  INTEGER ME, I, J, IT
  INTEGER ME_R, ME_C, ME_UP, ME_LEFT, ME_RIGHT,
ME_DOWN
  REAL A (0: NROW +1,0:NCOL+1)[0:LR-1,0:*]
  REAL B (1:NROW,1:NCOL)[0:LR-1,0:*]
  ME = THIS_IMAGE()
  ME_R = (ME-1)/LR
  ME_C = (ME-1) - ME_R*LR
  ME_UP = MOD(ME_R-1+LR,LR)
  ME_DOWN = MOD(ME_R+1+LR,LR)
  ME_RIGHT = MOD(ME_C+1+LC,LC)
  ME_LEFT = MOD(ME_C-1+LC,LC)
```

```

DO IT = 1, ITMAX
  CALL SYNC_ALL ( )
  DO J = 1, NCOL
    DO I = 1, NROW
      A(I, J) = B(I, J)
    ENDDO
  ENDDO

```

C  
C

Copying shadow elements of array A from  
neighbouring images before loop execution

```

A(1:NROW,NCOL+1) = A(1:NROW,1)[ME_R, ME_RIGHT]
A(1:NROW,0) = A(1:NROW,NCOL)[ME_R, ME_LEFT]
A(NROW+1,1:NCOL) = A(1,1:NCOL)[ME_DOWN,ME_C ]
A(0,1:NCOL) = A(NROW,1:NCOL)[ME_UP,ME_C ]

```

```

DO J = 1, NCOL
  DO I = 1, NROW
    B(I, J) = (A( I-1, J ) + A( I, J-1 ) +
*      A( I+1, J) + A( I, J+1 )) / 4
  ENDDO
ENDDO

```

```

ENDDO
CALL SYNC_ALL ( )
END

```

# Развитие языков (PGAS=>APGAS)

Добавление асинхронности в CAF и UPC

CAF 2.0:

- подгруппы процессов,
- топологии,
- новые средства синхронизации процессов,
- коллективные коммуникации,
- средства совмещения обменов с вычислениями,
- динамическое создание асинхронных активностей

(этого нет в Фортране 2008)

Тест	SEQ	MPI	DVM	CAF	MPI/ SEQ	DVM/ SEQ	CAF/ SEQ
BT	2593	3669	2604	3767	1.41	1.01	1.45
CG	506	1039	570	1086	2.05	1.12	2.14
EP	130	180	150	-	1.38	1.15	-
FT	704	1264	837	-	1.79	1.18	-
IS	428	665	590	-	1.55	1.37	-
LU	2731	3577	3112	-	1.31	1.13	-
MG	907	1637	1485	1784	1.80	1.63	1.96
SP	2124	3222	2452	3371	1.51	1.15	1.58
<b>Σ</b>	10123	15253	11800	-	1.50	1.16	-

**SEQ – последовательная программа, MPI –параллельная MPI-программа  
DVM – параллельная DVM-программа, CAF – параллельная CoArray Fortran  
программа**

# Создаваемые языки параллельного программирования (HPCS)

DARPA – High Productivity Computing Systems  
2002-2010

(эффективность, программируемость,  
переносимость ПО, надежность)

Chapel (Cray), APGAS

X10 (IBM), APGAS

ООП



# Цели разработки Chapel (Cray)

- Снизить трудоемкость программирования
  - написание программы, ее изменение, тюнинг, портирование, сопровождение
- Эффективность программ на уровне MPI
  - сравнимо на обычных кластерах
  - выше на более продвинутых архитектурах
- Эффективность при портировании
  - как MPI, но лучше OpenMP, CAF, UPC
- Надежность
  - исключить часто встречающиеся ошибки
  - повысить уровень абстракции чтобы снизить вероятность других ошибок

# Основные черты языка Chapel

- Полнота поддержки параллелизма
  - параллелизм по данным, параллелизм задач, вложенный параллелизм
  - отражение всех уровней параллелизма программы
  - использование всех уровней параллелизма ЭВМ
- Поддержка глобального (а не фрагментарного!) взгляда на программу (управление и данные)
- Управление локализацией данных и вычислений, задаваемые программистом методы распределения
- Включение широко используемых возможностей последовательных языков (ООП)
- Новый синтаксис (чтобы не путать с привычным!) и семантика конструкций языка

# Основные отличия X10 (IBM)

- Является расширением языка Java
- Отличие доступа к локальным и удаленным данным (как в языках PGAS)
- Свои методы синхронизации
- Ориентация на параллельные системы с распределенной памятью
- Open source проект

```

PROGRAM  JACOB_DVM_OpenMP
PARAMETER (L=4096, ITMAX=100)
REAL  A(L,L), B(L,L)
CDVM$  DISTRIBUTE  (BLOCK, BLOCK)  ::  A
CDVM$  ALIGN  B(I,J)  WITH  A(I,J)
PRINT *, '***** TEST_JACOBI *****'
C$OMP  PARALLEL
      DO IT = 1, ITMAX
CDVM$  PARALLEL (J,I) ON A(I, J)
C$OMP  DO
      DO J = 2, L-1
        DO I = 2, L-1
          A(I, J) = B(I, J)
        ENDDO
      ENDDO
CDVM$  PARALLEL (J,I) ON B(I, J), SHADOW_RENEW (A)
C$OMP  DO
      DO J = 2, L-1
        DO I = 2, L-1
          B(I, J) = (A(I-1, J) + A(I, J-1) + A(I+1, J) +
*          A(I, J+1)) / 4
        ENDDO
      ENDDO
ENDDO
C$OMP  END PARALLEL
END

```

```

PROGRAM JACOB_DVMH
PARAMETER (L=4096, ITMAX=100)
REAL A(L,L), B(L,L)
CDVM$ DISTRIBUTE (BLOCK, BLOCK) :: A
CDVM$ ALIGN B(I,J) WITH A(I,J)
PRINT *, '***** TEST_JACOBI *****'
DO IT = 1, ITMAX
CDVM$ REGION
CDVM$ PARALLEL (J,I) ON A(I, J)
    DO J = 2, L-1
        DO I = 2, L-1
            A(I, J) = B(I, J)
        ENDDO
    ENDDO
CDVM$ PARALLEL (J,I) ON B(I, J), SHADOW_RENEW (A)
    DO J = 2, L-1
        DO I = 2, L-1
            B(I, J) = (A(I-1, J) + A(I, J-1) + A(I+1, J) +
*           A(I, J+1)) / 4
        ENDDO
    ENDDO
CDVM$ END REGION
ENDDO
END

```

# Времена выполнения программы JASRED\_DVMH (сек) на кластере K-100

Число ядер	Serial	DVM	DVMH (32x16x1)	PGI_APM	Fortran CUDA (32x16x1)	OpenMP
<b>1</b>	<b>46.2</b>	<b>51.1</b>	<b>7.7</b>	<b>13.7</b>	<b>5.5</b>	<b>46.0</b>
<b>2</b>		<b>25.7</b>	<b>4.2</b>			<b>23.1</b>
<b>3</b>		<b>18.6</b>	<b>3.1</b>			<b>19.6</b>
<b>4</b>		<b>15.3</b>				<b>13.8</b>
<b>8</b>		<b>10.5</b>				<b>11.7</b>
<b>11</b>		<b>12.3</b>				<b>11.0</b>

# План изложения

- Модели и языки программирования с явным параллелизмом (абстрактная и целевая машина, уровень, компилятор/библиотека, новый язык/расширение существующего, новые конструкции/директивы компилятору)
- **Языки программирования с неявным параллелизмом + автоматическое распараллеливание (Fortran, C, C++)**
- Автоматизация преобразования имеющихся последовательных программ в эффективные параллельные программы на языках с явным или неявным параллелизмом
- Автоматизация отладки

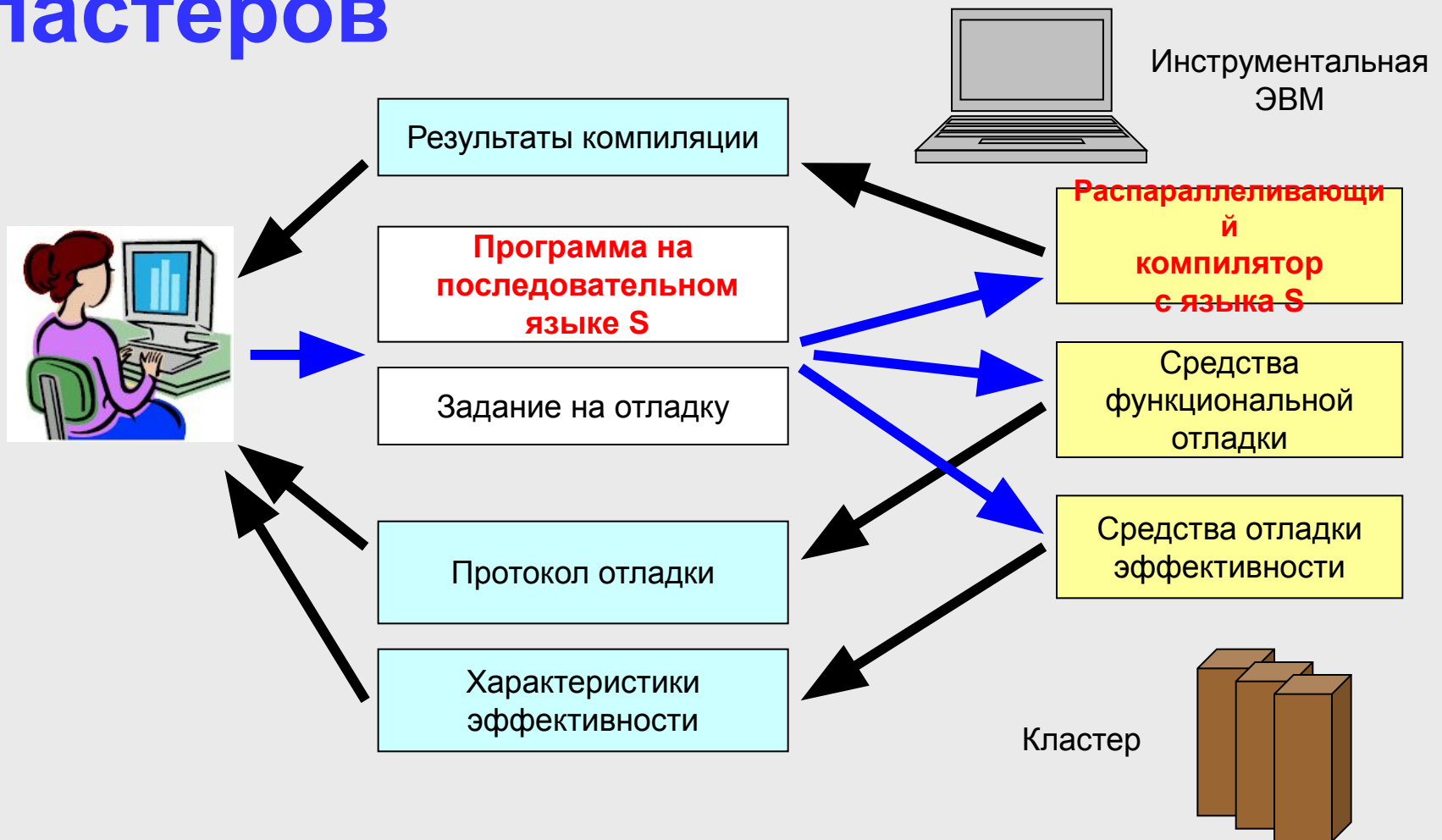
# Автоматизация параллельного программирования

Два новых (2005 г.) направления автоматизации в системе DVM:

- дисциплина написания на языках последовательного программирования **“потенциально параллельных программ”** - таких программ, которые могут быть автоматически (без участия пользователя) преобразованы в эффективные параллельные программы
- автоматизированное (с участием пользователя) преобразование последовательных программ в **потенциально параллельные программы**



# Автоматизация разработки параллельных программ для кластеров



# Алгоритм работы распараллеливающего компилятора АРК-DVM

## Анализ последовательной программы

Формирование вариантов распределения данных (ВРД)

Для каждого ВРД формирование схем распараллеливания - добавление вариантов распределения вычислений и доступа к данным

Оценка эффективности каждой схемы на множестве возможных решеток процессоров заданной ЭВМ и выбор наилучшей схемы

Генерация параллельной программы по лучшей схеме

# Анализ последовательной программы

Статический анализ – указатели, косвенная индексация, параметры....

Динамический анализ – ресурсы времени и памяти, представительность входных данных



Дополнительные спецификации программиста (декларативные, без ориентации на модель параллелизма), например, независимость витков, размеры массивов,....

# Алгоритм Якоби на языке Fortran

```
PROGRAM JACOB_SEQ
  PARAMETER (L=8, ITMAX=20)
  REAL A(L,L), B(L,L)

  PRINT *, '***** TEST_JACOBI *****'

  DO IT = 1, ITMAX

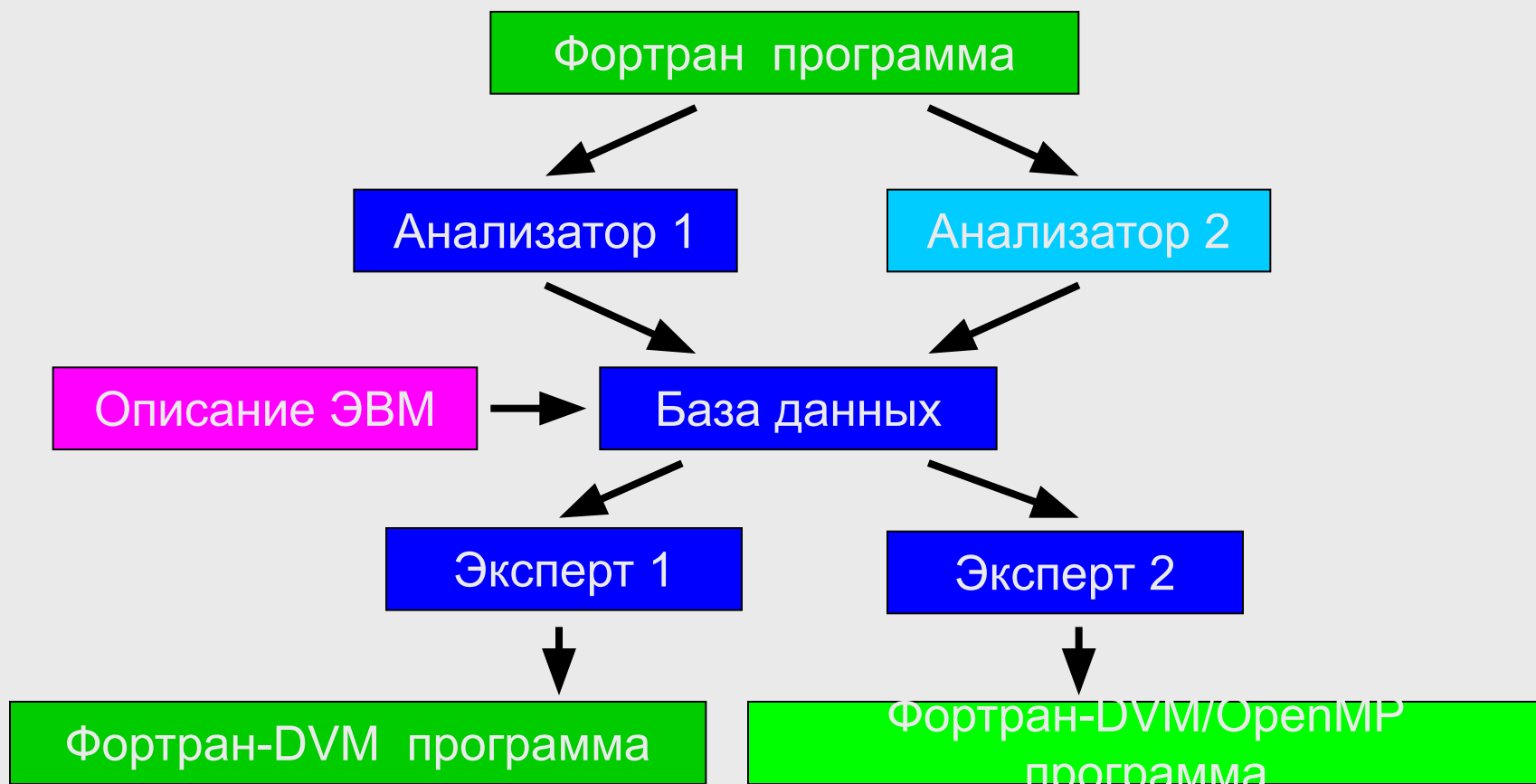
    DO J = 2, L-1
      DO I = 2, L-1
        A(I, J) = B(I, J)
      ENDDO
    ENDDO

    DO J = 2, L-1
      DO I = 2, L-1
        B(I, J) = (A(I-1, J) + A(I, J-1) + A(I+1, J) +
*                A(I, J+1)) / 4
      ENDDO
    ENDDO
  ENDDO
END
```

# Поиск наилучшей схемы распараллеливания

- Логическая сложность алгоритмов построения и оценки схем распараллеливания (мало зависит от выходного языка параллельного программирования  $P$ )
- Огромное число возможных схем  $\Rightarrow$  необходимы эвристики

# Блоки компилятора АРК-DVM



# Апробация компилятора АРК-DVM

Работа компилятора проверена на:

- тестах NAS LU, BT, SP (3D Навье-Стокс) - класс C и A
- программе MHPDV (трехмерного моделирования сферического взрыва во внешнем магнитном поле с помощью решения уравнений идеальной магнитогидродинамики)
- программе ZEBRA (расчет нейтронных полей атомного реактора в диффузионном приближении)

# Характеристики программ

	<b>BT</b>	<b>LU</b>	<b>SP</b>	<b>MHPDV</b>	<b>ZEBRA</b>
Количество строк	10442	3635	5950	1878	2426
Количество циклов	504	424	<b>499</b>	116	49
Количество циклов, распараллеленных DVM-экспертом	481	410	<b>293</b>	115	28
Количество массивов	34	30	37	33	40
Суммарное число измерений	75	64	70	78	49
Количество построенных схем	16	16	16	16	64



# Времена выполнения (сек) на МВС-100К DVM-программ класса С

Варианты программ	1 проц	8 проц	64 проц	256 проц	1024 проц
BT-авт	мало ОП	1255.97	182.70	54.64	<b>21.36</b>
BT-ручн	мало ОП	817.88	128.01	30.27	<b>7.19</b>
LU-авт	<b>3482.40</b>	1009.49	148.78	40.33	25.55
LU-ручн	<b>2103.14</b>	858.26	122.61	34.99	19.97
SP-авт	<b>1982.00</b>	-	-	-	-
SP-ручн	<b>2601.85</b>	-	-	-	-
MHPDV-авт	3703.23	500.78	89.32	34.75	12.78
MHPDV-ручн	3574.29	486.74	79.63	32.15	10.98
ZEBRA-авт	75.09	11.13	1.96	-	-
ZEBRA-ручн	75.62	10.18	1.85	-	-

# План изложения

- Модели и языки программирования с явным параллелизмом (абстрактная и целевая машина, уровень, компилятор/библиотека, новый язык/расширение существующего, новые конструкции/директивы компилятору)
- Языки программирования с неявным параллелизмом + автоматическое распараллеливание (**Fortran**, C, C++)
- **Автоматизация преобразования имеющихся последовательных программ в эффективные параллельные программы на языках с явным или неявным параллелизмом**
- Автоматизация отладки

# Автоматизированное распараллеливание последовательных программ

**Диалог с пользователем для:**

- Исследования результатов анализа и задания характеристик программы, недоступных для анализа но необходимых для распараллеливания
- Исследования предлагаемых вариантов распараллеливания и выбора из них наиболее предпочтительных

Пользователю предоставляются прогнозируемые характеристики эффективности (всей программы и ее отдельных циклов) для разных конфигураций интересующих его ЭВМ



Files and PU
luC_u.for
ludv2
read_input
domain
setcoeff
setbv
setiv
exact

Program units

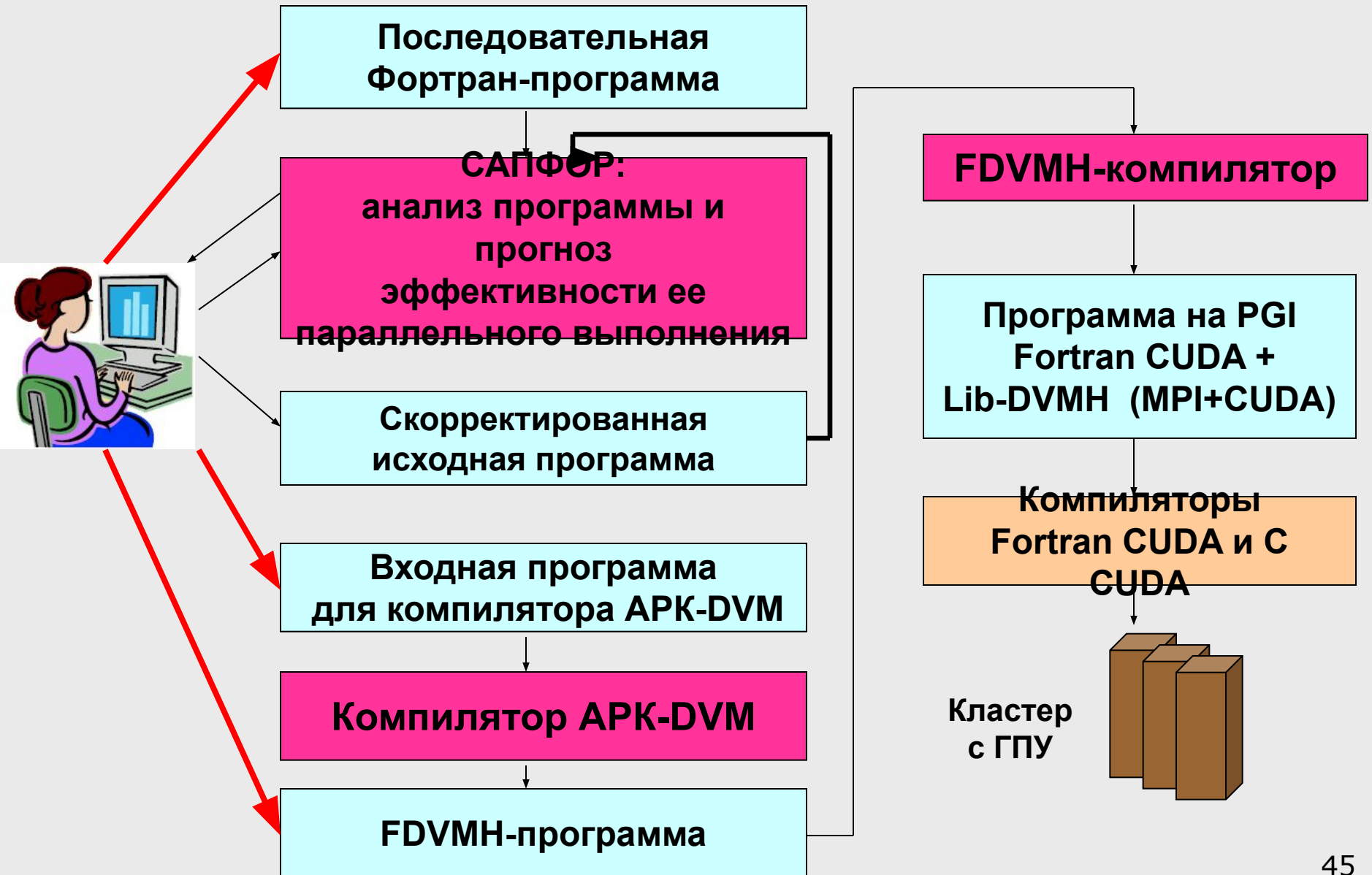
```
do k = nz-1,2,-1
  do j = jend, jst, -1
    do i = iend, ist, -1
```

```
    r43 = ( 4.0d+00 / 3.0d+00 )
    c1345 = c1 * c3 * c4 * c5
    c34 = c3 * c4
```

c  
c form the block daigonal

Loops / PU	Pro...	File name	Level ...	Seque...	Parall...	Speed...	Com...	Dependency	Read variab
k=2,nz-1	setiv	luC_u.for (594)	6.1	151.1...	19.128...	7.9024...	0.0000...		
l=1,isiz3	ssor	luC_u.for (1334)	9.1	106.2...	13.450...	7.9024...	0.0000...		
k=2,nz-1	blts	luC_u.for (2200)	11.1	65.53...	8.6304...	7.5935...	0.5590...	REG (rsd);	rsd(1,:j-1,k);
k=nz-1,2,-1	buts	luC_u.for (2797)	12.1	63.10...	8.7710...	7.1951...	0.1269...	REG (rsd);	rsd(1,:j+1,k);
k=2,nz0-1	l2no...	luC_u.for (3380)	13.2	61.43...	9.7717...	6.2875...	8.4581...		
k=2,nz-1	error	luC_u.for (3432)	14.2	45.05...	5.7729...	7.8046...	1.1276...		
k=1,nz	rhs	luC_u.for (1569)	10.1	42.51...	5.3800...	7.9024...	0.0000...		
k=2,nz-1	rhs	luC_u.for (1608)	10.3	40.95...	5.2480...	7.8048...	0.0000...		
k=2,nz-1	rhs	luC_u.for (1793)	10.12	40.95...	5.2480...	7.8048...	0.3179...		
k=2,nz-1	rhs	luC_u.for (1979)	10.21	40.95...	5.2480...	7.8048...	0.0630...		

# Схема использования DVMH-модели



# Принципиальные решения

## **DVM-подход:**

- Программист принимает основные решения по распараллеливанию, а компилятор и система поддержки обеспечивают их реализацию
- Спецификации параллелизма – это комментарии к последовательной программе

## **Автоматическое распараллеливание на кластер:**

- Подсказки пользователя о свойствах программы, выбор схемы распараллеливания посредством прогнозирования

## **Автоматизированное распараллеливание программ:**

- Автоматическое распараллеливание + взаимодействие с программистом на этапе выбора схемы распараллеливания

# План изложения

- Модели и языки программирования с явным параллелизмом
- Языки программирования с неявным параллелизмом + автоматическое распараллеливание (НОРМА, **Fortran**, **C**, **C++**)
- Автоматизация преобразования имеющихся последовательных программ в эффективные параллельные программы на языках с явным или неявным параллелизмом
- **Автоматизация отладки**

# Автоматизация функциональной отладки параллельных программ

- Автоматизированное сравнение поведения и промежуточных результатов разных версий последовательной программы
- Динамический анализ корректности параллельной программы (автоматический)
- Автоматическое сравнение поведения и промежуточных результатов параллельной и последовательной программ



# Функциональная отладка DVM-программ

Используется следующая методика поэтапной отладки программ:

- На первом этапе программа отлаживается на рабочей станции как последовательная программа, используя обычные методы и средства отладки
- На втором этапе программа выполняется на той же рабочей станции в специальном режиме проверки распараллеливающих указаний
- На третьем этапе программа выполняется на рабочей станции или на параллельной машине в специальном режиме, когда промежуточные результаты параллельного выполнения сравниваются с эталонными результатами (например, результатами последовательного выполнения)

# Типы ошибок

- Синтаксические ошибки в DVM-указаниях и нарушение статической семантики
- Неправильная последовательность выполнения DVM-указаний или неправильные параметры указаний
- Неправильное выполнение вычислений из-за некорректности DVM-указаний и ошибок, не проявляющихся при последовательном выполнении программы
- Аварийное завершение параллельного выполнения программы (авосты, зацикливания, зависания) из-за ошибок, не проявляющихся при последовательном выполнении программы

# Динамический контроль

- Чтение неинициализированных переменных
- Выход за пределы массива
- Необъявленная зависимость по данным в параллельной конструкции
- Модификация в параллельной ветви размноженных переменных (не редукционных и не приватных)
- Необъявленный доступ к нелокальным элементам распределенного массива
- Чтение теневых граней распределенного массива до завершения операции их обновления
- Использование редукционных переменных до завершения операции асинхронной редукции

# Сравнение результатов

- Где и что сравнивать
- Две трассы, одна трасса, без трасс – на лету
- Получение эталонной трассировки - управление объемом при компиляции, через параметры запуска, с помощью файла конфигурации
- Представительные витки циклов
- Особенности сравнения (редукция, учет правила собственных вычислений, точность)

# Автоматизация отладки эффективности параллельных программ

- Прогноз характеристик эффективности каждого варианта распараллеливания для разных конфигураций интересующих пользователя ЭВМ
- Прогноз характеристик эффективности полученной параллельной программы для разных конфигураций интересующих пользователя ЭВМ
- Получение реальных характеристик эффективности параллельной программы при ее запусках на разных конфигурациях ЭВМ
- Автоматическое сравнение прогнозируемых и реальных характеристик

# Эффективность выполнения параллельной программы

Эффективность выполнения параллельной программы выражается в ускорении вычислений, что может привести:

- к сокращению времени выполнения задачи фиксированного размера;
- к росту размера задачи, которую удастся решить за то же самое время.

# Критерии эффективности выполнения параллельных программ

- **Ускорение**  $S_n = T_1 / T_n$ ,  
где  $T_n$  — время исполнения параллельной программы на  $n$  процессорах,  $T_1$  — время ее выполнения на 1-ом процессоре (или время выполнения исходной последовательной программы, или прогнозируемое время выполнения на 1 процессоре параллельной программы).  
В идеальном случае (отсутствие накладных расходов на организацию параллелизма), как может показаться, ускорение равно  $n$  (линейное ускорение).
- **Коэффициент эффективности**  $S_n / n = T_1 / nT_n$ . В идеальном случае, как может показаться, он равен 1, или 100 %.

# Критерии эффективности выполнения параллельных программ

- Если ускорение линейно от  $n$ , то говорят, что такая программа обладает свойством **масштабируемости** (возможностью ускорения вычислений пропорционально числу процессоров).
- Особое внимание заслуживает случай  $Sn > n$  (**суперлинейное ускорение**).
- Требования исходной задачи могут превосходить возможности используемого процессора по его ресурсам (чаще всего это кеши различных уровней, буфер TLB, ОП). При запуске на нескольких процессорах на них попадают подзадачи меньшего объёма, и требования к объёму ресурсов процессора сокращаются. При преодолении таких порогов и возникает суперлинейное ускорение.



# Факторы, определяющие эффективность выполнения параллельных программ

- степень распараллеливания программы - доля параллельных вычислений в общем объеме вычислений
- равномерность загрузки процессоров во время выполнения параллельных вычислений
- время ожидания межпроцессорных обменов (зависит от рассинхронизации процессов и степени совмещения межпроцессорных обменов с вычислениями)
- эффективностью выполнения вычислений на каждом процессоре (кеш, подкачка страниц ВП, расходы на организацию параллелизма, выполнение процессором других программ)

# Характеристики эффективности параллельных программ

Пользователь может получить следующие характеристики эффективности программы и отдельных ее частей:

- **execution time** - астрономическое время выполнения
- **productive time** - прогнозируемое время выполнения на одном процессоре
- **parallelization efficiency** – прогнозируемая эффективность параллельного выполнения =  
$$\text{productive time} / (N * \text{execution time})$$
- **lost time** – потерянное время =  
$$N * \text{execution time} - \text{productive time}$$
  
где N - число процессоров

# Характеристики эффективности параллельных программ

Компоненты **lost time**:

- **insufficient parallelism** - потери из-за выполнения последовательных частей программы на одном или всех процессорах
- **communication** - потери из-за межпроцессорных обменов или синхронизаций
- **Idle time** - простои процессоров из-за отсутствия работы

и важный компонент времени коммуникаций –

**real synchronization** - реальные потери из-за рассинхронизации

# Характеристики эффективности параллельных программ

Кроме того, выдаются характеристики:

- **load imbalance** - возможные потери из-за разной загрузки процессоров
- **synchronization** - возможные потери на синхронизацию
- **time\_variation** - возможные потери из-за разброса времен
- **overlap** - возможное сокращение коммуникационных расходов за счет совмещения межпроцессорных обменов с вычислениями

# Подходы к вычислению и выдаче характеристик

## Как собирать

- трассировка (для каждого процессора, каждой нити)
- статистика
- количество обращений \* прогнозируемое время

## Как выдавать

- статистика по интервалам
- статистика по конструкциям языка
- временные диаграммы

# Проблема – нестабильность характеристик

## Нестабильность коммуникаций

- Изменение состава процессоров при неоднородности коммуникационной среды
- Загрузка коммуникационной среды другими работами

Можно выдавать стабильные характеристики коммуникаций (вычислять их по формулам, зависящим от длин сообщений, латентности и пропускной способности коммуникационной среды)

# Нестабильность производительности процессоров

- **Попадание на медленные процессоры**  
(появляется разбалансировка, можно запрашивать лишние процессоры и отключать медленные)
- **Частая активизация системных процессов**  
(возрастает время коммуникаций за счет времени реальной рассинхронизации)

Можно моделировать не только коммуникации, но и загрузку процессоров

=> предсказание эффективности

# Предсказатель эффективности DVM-программ

получает характеристики выполнения DVM-программы на рабочей станции и использует их для предсказания эффективности ее выполнения на кластере с заданными параметрами (конфигурация, производительность процессоров, количество и характеристики каналов связи)

Для реализации такой схемы предсказания необходимо тщательное проектирование интерфейса с run-time системой



# Принципиальные трудности предсказания эффективности

- Для современных процессоров трудно прогнозировать время выполнения разных фрагментов программы (кэш-память и динамическое планирование выполнения)
- Трудно моделировать работу программных компонентов коммуникационной системы

= > очень сложно получить точные характеристики выполнения программы

# Предсказатель – инструмент отладки эффективности

Он может довольно точно оценить влияние основных факторов:

- степень распараллеливания программы - доля параллельных вычислений в общем объеме вычислений
  - равномерность загрузки процессоров во время выполнения параллельных вычислений
  - время, необходимое для выполнения межпроцессорных обменов (и синхронизаций)
  - степень совмещения межпроцессорных обменов с вычислениями
- => есть еще важный фактор – эффективное выполнение вычислений на процессорах

# Предсказатель – инструмент отладки эффективности

На современных процессорах эффективность вычислений может отличаться в 3-7 раз в зависимости от их согласованности с организацией кэш-памяти

Поэтому важно предоставить программисту инструмент, помогающий ему обеспечить такую согласованность.

Согласованность же с особенностями коммуникационных сетей (группировка или разбиение сообщений, их планирование) должны обеспечивать системы программирования с языков высокого уровня

# Выводы

- Отладка эффективности параллельных программ – процесс очень сложный и трудоемкий
- Развитые средства анализа эффективности могут существенно ускорить этот процесс
- Из-за нестабильности характеристик эффективности при коллективном использовании параллельных ЭВМ важную роль могут сыграть средства предсказания этих характеристик посредством моделирования выполнения параллельных программ
- Для достижения эффективности параллельной программы приходится многократно изменять программу, иногда кардинально меняя схему ее распараллеливания. Поэтому важно использовать высокоуровневые средства разработки параллельных программ

**Вопросы, замечания?**

**СПАСИБО !**

# Информация о DVM-системе

- Система (в виде исходных текстов и библиотек выполняемых программ) доступна через Интернет (<http://www.keldysh.ru/dvm/>)
- Вместе с системой поставляется набор демонстрационных DVM-программ, а также документация пользователя и разработчика