Автоматизация создания вычислительных программ для современных кластеров

В.А. Крюков

Институт прикладной математики им. М.В. Келдыша РАН

krukov@keldysh.ru

31.03.2011

План изложения

- Модели и языки программирования с явным параллелизмом (**MPI**, Shemem, Pthreads, **CUDA-**OpenCL, RCCE-MCAPI, HPF, **DVM**, CAF, **OpenMP**, PGI_APM, Chapel, X10, Fortress, **DVM/OpenMP, DVM/PGI_APM**)
- Языки программирования с неявным параллелизмом + автоматическое распараллеливание (HOPMA, **Fortran**, C, C++)
- Автоматизация преобразования имеющихся последовательных или MPI-программ в эффективные параллельные программы на языках с явным или неявным параллелизмом

Модели и языки с явным параллелизмом

<table>
<tr><td colspan="5">Chapel, X10, Fortress</td></tr>
<tr><td colspan="4">DVM/OpenMP/PGI_APM</td><td rowspan="3"></td></tr>
<tr><td colspan="3">DVM/OpenMP</td><td></td></tr>
<tr><td colspan="2">HPF, DVM, CAF-UPC</td><td>OpenMP</td><td>PGI_APM</td></tr>
<tr><td>MPI</td><td>Shemem</td><td>Pthreads</td><td>CUDA-
OpenCL</td><td>RCCE-
MCAPI</td></tr>
</table>

Алгоритм Якоби на языке Fortran

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

```
      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
```

```
      PROGRAM    JAC_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$OMP PARALLEL SHARED(A,B) PRIVATE (I,J,IT)
      DO IT  =  1,  ITMAX
CDVM$    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
```

```
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
```

```
      PROGRAM    JAC_DVM_PGI_APM
      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   **********'
!$acc data region copyout (B), local (A)
      DO IT  =  1,  ITMAX
!$acc 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
!$acc end  region
         ENDDO
!$acc end data region
      END
```

План изложения

- Модели и языки программирования с явным параллелизмом (**MPI**, Shemem, Pthreads, **CUDA-**OpenCL, RCCE-MCAPI, HPF, **DVM**, CAF, **OpenMP**, PGI_APM, Chapel, X10, Fortress, **DVM/OpenMP**, **DVM/PGI_APM**)
- **Языки программирования с неявным параллелизмом + автоматическое распараллеливание** (НОРМА, **Fortran**, C, C++)
- Автоматизация преобразования имеющихся последовательных или MPI-программ в эффективные параллельные программы на языках с явным или неявным параллелизмом

Автоматизация параллельного программирования

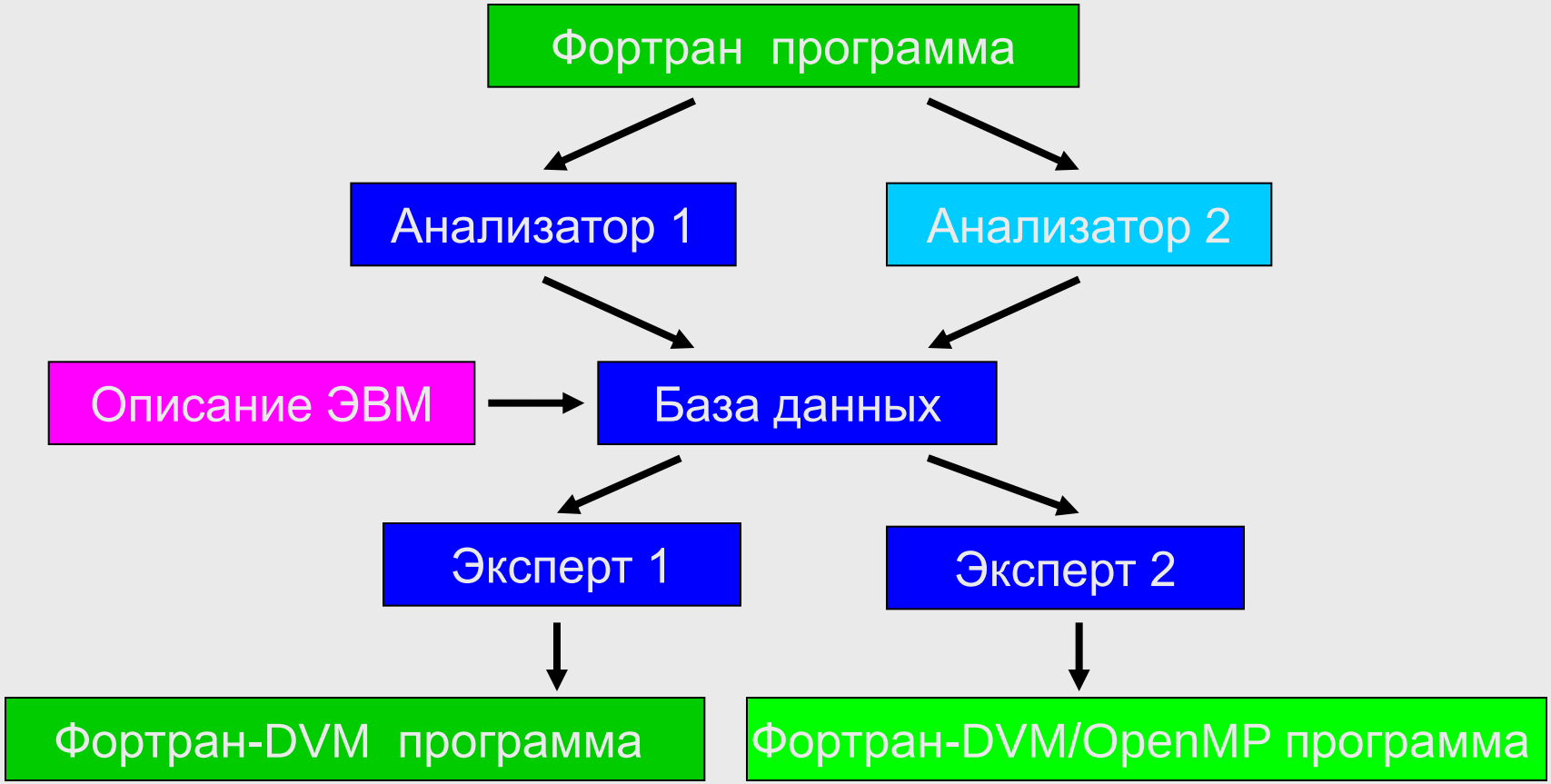
Два новых (2005 г.) направления автоматизации в системе DVM:

- 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**

Использование языков с неявным параллелизмом

<table>
<tr><td colspan="5">Fortran, C, C++</td></tr>
<tr><td colspan="4">DVM/OpenMP/PGI_APM</td><td rowspan="3"></td></tr>
<tr><td colspan="3">DVM/OpenMP</td><td></td></tr>
<tr><td colspan="2">HPF, DVM, CAF-UPC</td><td>OpenMP</td><td>PGI_APM</td></tr>
<tr><td>MPI</td><td>Shemem</td><td>Pthreads</td><td>CUDA-
OpenCL</td><td>RCCE-
MCAPI</td></tr>
</table>

Схема работы автоматически распараллеливающего 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

	BT	LU	SP	MHPDV	ZEBRA
Количество строк	10442	3635	5950	1878	2426
Количество циклов	504	424	**499**	116	49
Количество циклов, распараллеленных DVM-экспертом	481	410	**293**	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	**21.36**
BT-ручн	мало ОП	817.88	128.01	30.27	**7.19**
LU-авт	**3482.40**	1009.49	148.78	40.33	25.55
LU-ручн	**2103.14**	858.26	122.61	34.99	19.97
SP-авт	**1982.00**	-	-	-	-
SP-ручн	**2601.85**	-	-	-	-
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	-	-

Времена (класс A) на 1 узле СКИФ-МГУ

Варианты программ	**1 ядро**	**2 ядра**	**4 ядра**	**8 ядер**
BT-авт	119.04	61.39	39.56	**34.66**
BT-ручн	109.78	55.88	37.65	**34.73**
LU-авт	110.42	60.56	39.01	**33.35**
LU-ручн	106.58	57.09	37.88	**35.53**
SP-авт	81.08	76.64	75.42	**76.62**
SP-ручн	102.82	51.42	33.19	**32.91**
MHPDV-авт	300.19	148.87	82.89	**51.60**
MHPDV-ручн	333.80	169.97	91.25	**57.55**
ZEBRA-авт	45.91	21.85	11.71	**5.92**
ZEBRA-ручн	53.68	26.17	14.72	**8.11**

План изложения

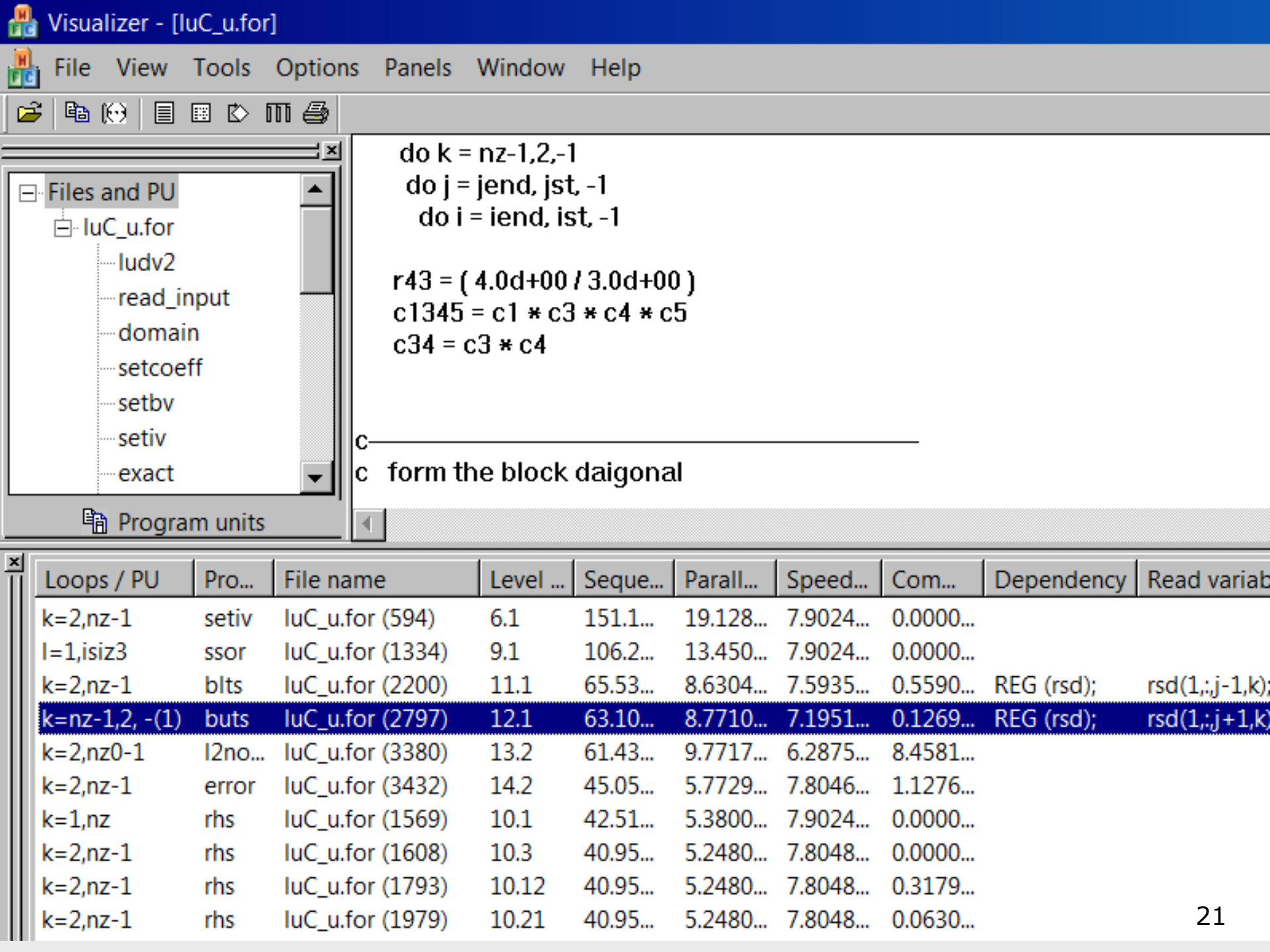
- Модели и языки программирования с явным параллелизмом (**MPI**, Shemem, Pthreads, **CUDA-**OpenCL, RCCE-MCAPI, HPF, **DVM**, CAF, **OpenMP**, PGI_APM, Chapel, X10, Fortress, **DVM/OpenMP, DVM/PGI_APM**)
- Языки программирования с неявным параллелизмом + автоматическое распараллеливание (HOPMA, **Fortran**, C, C++)
- **Автоматизация преобразования имеющихся последовательных или MPI-программ в эффективные параллельные программы на языках с явным или неявным параллелизмом**

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
- Преобразования исходной программы

Пользователю предоставляются прогнозируемые характеристики эффективности (всей программы и ее отдельных циклов)


Visualizer - [luC_u.for]
File View Tools Options Panels Window Help
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...

Вопросы, замечания?

СПАСИБО !