Суперкомпьютеры и параллельная обработка данных

Бахтин Владимир Александрович
к.ф.-м.н., ведущий научный сотрудник
Института прикладной математики им М.В.Келдыша РАН
кафедра системного программирования
факультет вычислительной математики и кибернетики
Московского университета им. М.В. Ломоносова

Internal Control Variables. stack-size-var

```
int main () {
   int a[1024][1024];
   #pragma omp parallel private (a)
   {
      for (int i=0;i<1024;i++)
        for (int j=0;j<1024;j++)
          a[i][j]=i+j;
   }
}
```

icl /Qopenmp test.cpp

⇒Program Exception – stack overflow

Linux: ulimit -a

ulimit -s <stacksize in Kbytes>

Windows: /F<stacksize in bytes>

-Wl,--stack, <stacksize in bytes>

setenv KMP_STACKSIZE 10m

setenv GOMP_STACKSIZE 10000

setenv OMP_STACKSIZE 10M

Internal Control Variables. wait-policy-var

Подсказка OpenMP-компилятору о желаемом поведении нитей во время ожидания.
Начальное значение: зависит от реализации.
Существует одна копия этой переменной для всей программы.

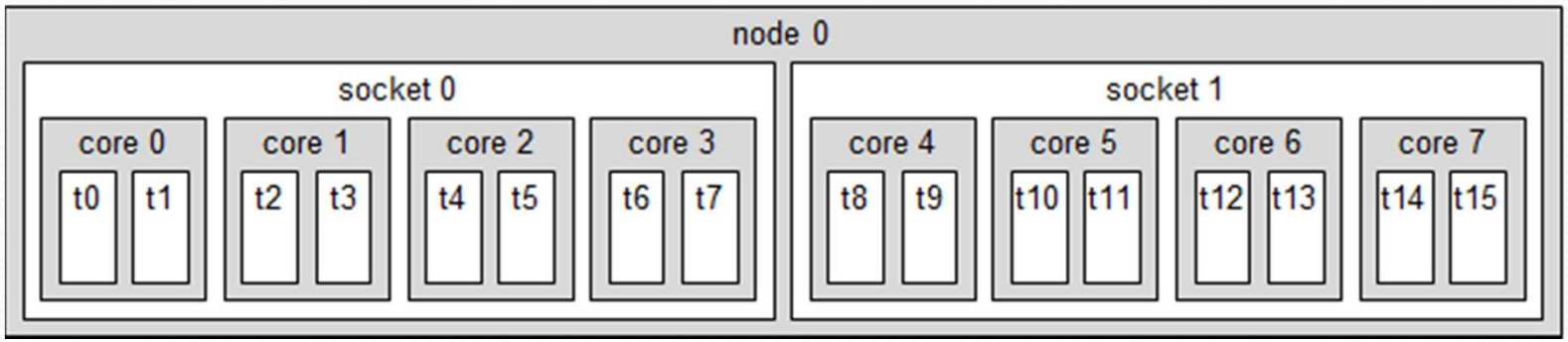
Значение переменной можно изменить:

setenv OMP_WAIT_POLICY ACTIVE
setenv OMP_WAIT_POLICY active
setenv OMP_WAIT_POLICY PASSIVE
setenv OMP_WAIT_POLICY passive

IBM AIX
SPINLOOPTIME=100000
YIELDLOOPTIME=40000

Internal Control Variables. place-partition-var

Распределение OpenMP-нитей по ядрам.
Начальное значение: зависит от реализации.
Существует одна копия этой переменной для всей программы.



setenv OMP_PLACES "threads(2)"
setenv OMP_PLACES "{0,1},{2,3},{4,5},{6,7},{8,9},{10,11},{12,13},{14,15}"
setenv OMP_PLACES "{0:2},{2:2},{4:2} ,{6:2},{8:2},{10:2},{12:2},{14:2}"
setenv OMP_PLACES "{0:2}:8:2"

Internal Control Variables. Приоритеты

клауза	вызов функции	переменная окружения	ICV
	omp_set_dynamic()	OMP_DYNAMIC	*dyn-var*
	omp_set_nested()	OMP_NESTED	*nest-var*
num_threads	omp_set_num_threads()	OMP_NUM_THREADS	*nthreads-var*
schedule	omp_set_schedule()	OMP_SCHEDULE	*run-sched-var*
schedule			*def-sched-var*
		OMP_STACKSIZE	*stacksize-var*
		OMP_WAIT_POLICY	*wait-policy-var*
		OMP_THREAD_LIMIT	*thread-limit-var*
	omp_set_max_active_ levels()	OMP_MAX_ACTIVE_ LEVELS	*max-active-levels-var*

Система поддержки выполнения OpenMP-программ

int omp_get_num_threads(void);
-возвращает количество нитей в текущей параллельной области

```
#include <omp.h>
void work(int i);
void test()
{
    int np;
    np = omp_get_num_threads(); /* np == 1*/
    #pragma omp parallel private (np)
    {
        np = omp_get_num_threads();
        #pragma omp for schedule(static)
        for (int i=0; i < np; i++)
            work(i);
     }
}
```

Система поддержки выполнения OpenMP-программ

int omp_get_thread_num(void);
-возвращает номер нити в группе [0: omp_get_num_threads()-1]

```
#include <omp.h>
void work(int i);
void test()
{
    int iam;
    iam = omp_get_thread_num(); /* iam == 0*/
    #pragma omp parallel private (iam)
    {
       iam = omp_get_thread_num();
       work(iam);
     }
}
```

Система поддержки выполнения OpenMP-программ

int omp_get_num_procs(void);
-возвращает количество процессоров, на которых программа выполняется

```
#include <omp.h>
void work(int i);
void test()
{
    int nproc;
    nproc = omp_get_num_ procs();
   #pragma omp parallel num_threads(nproc)
    {
       int iam = omp_get_thread_num();
       work(iam);
     }
}
```

Система поддержки выполнения OpenMP-программ

Распределение OpenMP-нитей по ядрам.

```
int omp_get_num_places(void);
int omp_get_place_num(void);

int main()
{
    int n_sockets, socket_num;
    omp_set_nested(1);                  // or export OMP_NESTED=true
    omp_set_max_active_levels(2);   // or export OMP_MAX_ACTIVE_LEVELS=2
    n_sockets = omp_get_num_places();
    #pragma omp parallel num_threads(n_sockets) private(socket_num)
    {
            socket_num = omp_get_place_num();
            socket_init(socket_num);
    }
}
```

Система поддержки выполнения OpenMP-программ

Распределение OpenMP-нитей по ядрам.

```
int omp_get_place_num_procs(int place_num);

void socket_init(int socket_num)
{
    int n_procs;
    n_procs = omp_get_place_num_procs(socket_num);
    #pragma omp parallel num_threads(n_procs) proc_bind(close)
    {
            printf("Reporting in from socket num, thread num:  %d %d\n",
            socket_num,omp_get_thread_num() );
    }
}


void omp_get_place_proc_ids(int place_num, int  *ids);
```

Система поддержки выполнения OpenMP-программ

int omp_get_level(void)
- возвращает уровень вложенности для текущей параллельной области.

```
#include <omp.h>
void work(int i) {
 #pragma omp parallel
    {
       int ilevel = omp_get_level ();
    }
}
void test()
{
     int ilevel = omp_get_level (); /*ilevel==0*/
    #pragma omp parallel private (ilevel)
    {
        ilevel =  omp_get_level ();
        int iam = omp_get_thread_num();
        work(iam);
      }
}
```

Система поддержки выполнения OpenMP-программ

int omp_get_active_level(void)

- возвращает количество активных параллельных областей (выполняемых 2-мя или более нитями).

```
#include <omp.h>
void work(int iam, int size) {
 #pragma omp parallel
    {
       int ilevel = omp_get_active_level ();
    }
}
void test()
{
    int size = 0;
    int ilevel = omp_get_active_level (); /*ilevel==0*/
    scanf("%d",&size);
    #pragma omp parallel if (size>10)
    {
       int iam = omp_get_thread_num();
        work(iam, size);
     }
}
```

Система поддержки выполнения OpenMP-программ

int omp_get_ancestor_thread_num (int level)

- для нити, вызвавшей данную функцию, возвращается номер нити-родителя, которая создала указанную параллельную область.

omp_get_ancestor_thread_num (0) = 0

```
If (level==omp_get_level()) {
    omp_get_ancestor_thread_num (level) == omp_get_thread_num ();
}

If ((level<0)||(level>omp_get_level())) {
    omp_get_ancestor_thread_num (level) == -1;
}
```

Система поддержки выполнения OpenMP-программ

int omp_get_team_size(int level);
- количество нитей в указанной параллельной области.

omp_get_team_size (0) = 1

```
If (level==omp_get_level()) {
    omp_get_team_size (level) == omp_get_num _threads ();
}

If ((level<0)||(level>omp_get_level())) {
    omp_get_team_size (level) == -1;
}
```

Система поддержки выполнения OpenMP-программ. Функции работы со временем

double omp_get_wtime(void);

возвращает для нити астрономическое время в секундах, прошедшее с некоторого момента в прошлом. Если некоторый участок окружить вызовами данной функции, то разность возвращаемых значений покажет время работы данного участка. Гарантируется, что момент времени, используемый в качестве точки отсчета, не будет изменен за время выполнения программы.

```
double start;
double end;
start = omp_get_wtime();
/*... work to be timed ...*/
end = omp_get_wtime();
printf("Work took %f seconds\n", end - start);
```

double omp_get_wtick(void);

- возвращает разрешение таймера в секундах (количество секунд между последовательными импульсами таймера).

Содержание

Новые возможности OpenMP

- Векторизация кода
- Обработка исключительных ситуаций / cancellation constructs
- Поддержка ускорителей/сопроцессоров

Использование векторизации

```
void add_float (float *a, float *b, float *c, float *d, float *e, int n)
{
    for (int i=0; i<n; i++)
      a[i] = a[i] + b[i] + c[i] + d[i] + e[i];
}
```

Использование векторизации

```
void add_float (float *restrict a, float *restrict b, float *restrict c,
float *restrict d, float *restrict e, int n) // C99
{
    for (int i=0; i<n; i++)
      a[i] = a[i] + b[i] + c[i] + d[i] + e[i];
}
```

Использование векторизации. Спецификация simd

#pragma omp simd *[clause[[,] clause]..]*
for-loops

#pragma omp for simd *[clause[[,] clause]..]*
for-loops

где клауза одна из:

- **safelen (length)**
- **linear (list[:linear-step])**
- **aligned (list[:alignment])**
- **private (list)**
- **lastprivate (list)**
- **reduction (reduction-identifier: list)**
- **collapse (n)**

Использование векторизации

```
void add_float (float *a, float *b, float *c, float *d, float *e, int n)
{
  #pragma omp simd
  for (int i=0; i<n; i++)
    a[i] = a[i] + b[i] + c[i] + d[i] + e[i];
}
```

```
void add_float (float *restrict a, float *restrict b, float *restrict c,
float *restrict d, float *restrict e, int n) // C99
{
   for (int i=0; i<n; i++)
     a[i] = a[i] + b[i] + c[i] + d[i] + e[i];
}
```

Использование векторизации. Спецификация declare simd

#pragma omp declare simd *[clause[[,] clause]..]*
function definition or declaration

где клауза одна из:

- **simdlen (length)**
  the largest size for a vector that the compiler is free to assume
- **linear (argument-list[:constant-linear-step])**
  in serial execution parameters are incremented by steps (induction variables with constant stride)
- **aligned (argument-list[:alignment])**
  all arguments in the argument-list are aligned on a known boundary not less than the specified alignment
- **uniform (argument-list)**
  shared, scalar parameters are broadcasted to all iterations
- **inbranch**
- **notinbranch**

Использование векторизации

```
#pragma omp declare simd notinbranch
float min(float a, float b) {
   return a < b ? a : b;
}

#pragma omp declare simd inbranch
float distance (float x, float y) {
   return (x - y) * (x - y);
}

#pragma omp parallel for simd
for (i=0; i<N; i++)
   d[i] = min (distance (a[i], b[i]), c[i]);
```

Cancellation Constructs

Директива

#pragma omp cancel *clause[[,] clause]*
где clause одна из:

- **parallel**
- **sections**
- **for**
- **taskgroup**
- **if (***scalar-expression***)**

Директива

#pragma omp cancellation point *clause[[,] clause]*
где clause одна из:

- **parallel**
- **sections**
- **for**
- **taskgroup**

Новая функция системы поддержки:

- **omp_get_cancellation**

Новая переменная окружения:

- **OMP_CANCELLATION**

Обработка исключительных ситуаций

```
void example() {
   std::exception *ex = NULL;
   #pragma omp parallel shared(ex)
   {
      #pragma omp for schedule(runtime)
      for (int i = 0; i < N; i++) {
         try {
            causes_an_exception();
         } catch (std::exception *e) {
            #pragma omp atomic write
               ex = e; // still must remember exception for later handling
            #pragma omp cancel for // cancel worksharing construct
         }
      }
      if (ex) { // if an exception has been raised, cancel parallel region
        #pragma omp cancel parallel
      }
   }
   if (ex) {  // handle exception stored in ex
   }
}
```

Поиск в дереве (часть 1)

```
typedef struct binary_tree_s {
   int value;
   struct binary_tree_s *left, *right;
} binary_tree_t;

binary_tree_t *search_tree_parallel (binary_tree_t *tree, int value) {
    binary_tree_t *found = NULL;
#pragma omp parallel shared(found, tree, value)
    {
#pragma omp taskgroup
        {
#pragma omp master
            {
                found = search_tree(tree, value, 0);
            }
        }
    }
    return found;
}
```

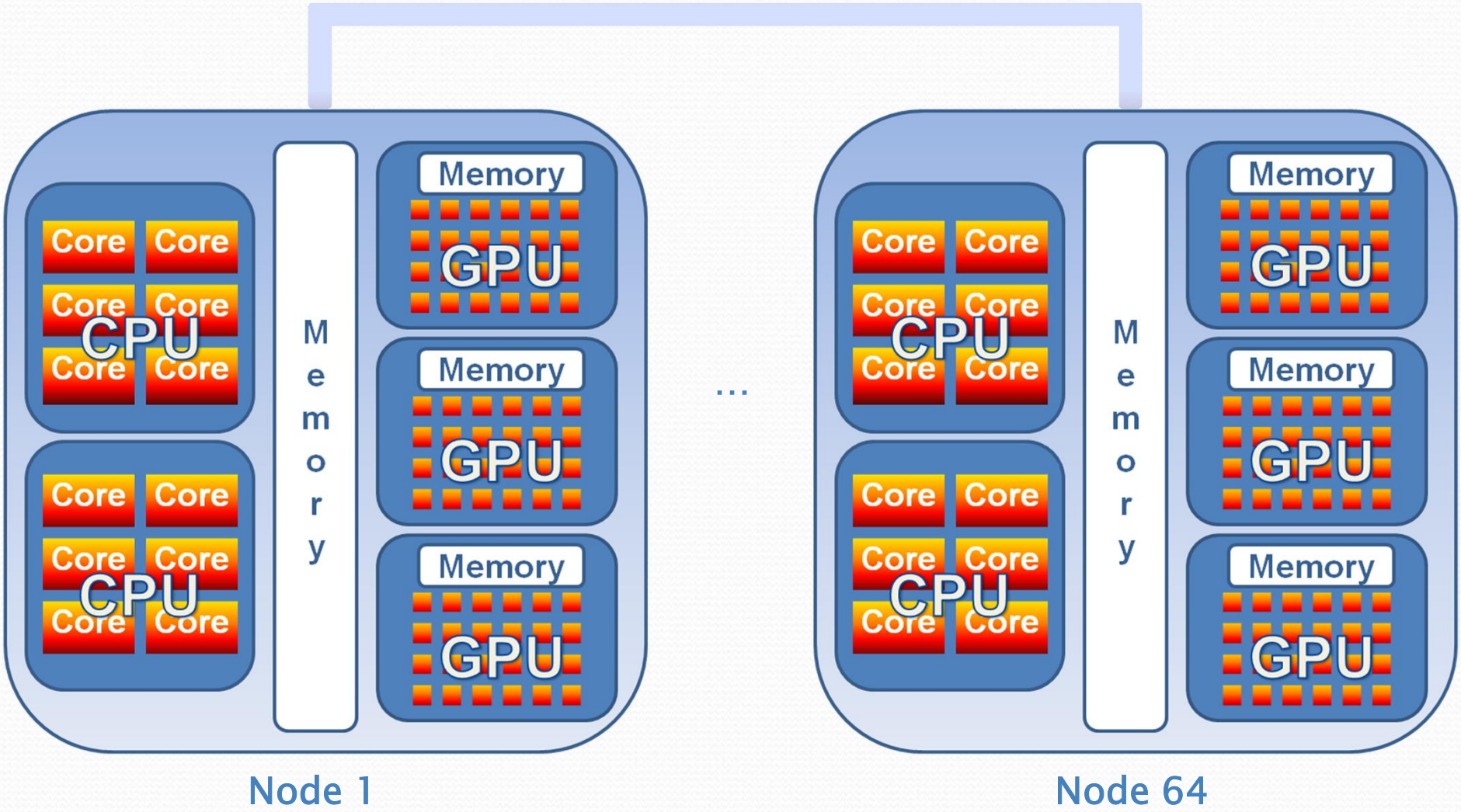
Поиск в дереве (часть 2)

```
binary_tree_t *search_tree(binary_tree_t *tree, int value, int level) {
    binary_tree_t *found = NULL;
    if (tree) {
        if (tree->value == value) {
            found = tree;
        } else {
            #pragma omp task shared(found) if(level < 10)
            {
                binary_tree_t *found_left = NULL;
                found_left = search_tree(tree->left, value, level + 1);
                if (found_left) {
                    #pragma omp atomic write
                      found = found_left;
                    #pragma omp cancel taskgroup
                }
            }
```

Поиск в дереве (часть 3)

```
            #pragma omp task shared(found) if(level < 10)
            {
                binary_tree_t *found_right = NULL;
                found_right = search_tree(tree->right, value, level + 1);
                if (found_right) {
                    #pragma omp atomic write
                        found = found_right;
                    #pragma omp cancel taskgroup
                }
            }
            #pragma omp taskwait
        }
    }
    return found;
}
```

Расширение OpenMP для использования ускорителей



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
```

Алгоритм Якоби на языке Fortran Cuda

```
PROGRAM JACOB_CUDA
      use cudafor
      use jac_cuda
      PARAMETER (L=4096,  ITMAX=100)
      parameter (block_dim = 16)
      real, device, dimension(l, l) :: a, b
      type(dim3) :: grid, block
      PRINT *, '*****  TEST_JACOBI   *******'
      grid = dim3(l / block_dim, l / block_dim, 1)
      block = dim3(block_dim, block_dim, 1)
      DO IT = 1, ITMAX
         call arr_copy<<<grid, block>>>(a, b, l)
         call arr_renew<<<grid, block>>>(a, b, l)
      ENDDO
END
```

Алгоритм Якоби на языке Fortran Cuda

```
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) A(I, J) = B(I, J)
      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)  B(I,J) =(A( I-1,J)+A(I,J-1)+A(I+1,J)+
A(I,J+1))/4
      end subroutine arr_renew
end module jac_cuda
```

Алгоритм Якоби в модели HMPP

```
!$HMPP jacoby codelet, target = CUDA
      SUBROUTINE JACOBY(A,B,L)
      IMPLICIT NONE
      INTEGER, INTENT(IN) :: L
      REAL, INTENT(IN) :: A(L,L)
      REAL, INTENT(INOUT) :: B(L,L)
      INTEGER I,J
      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
      END SUBROUTINE JACOBY
```

```
      PROGRAM JACOBY_HMPP
      PARAMETER    (L=4096,  ITMAX=100)
      REAL     A(L,L), B(L,L)
      PRINT *,  '**********TEST_JACOBI**********'
      DO IT  =  1,  ITMAX
!$HMPP    jacoby callsite
        CALL JACOBY(A,B,L)
      ENDDO
      PRINT *, B
      END
```

Алгоритм Якоби в модели HMPP

```
          PROGRAM    JACOBY_HMPP
           PARAMETER    (L=4096,  ITMAX=100)
           REAL     A(L,L), B(L,L)
!$hmpp jac allocate, args[A;B].size={L,L}
!$hmpp jac advancedload, args[B]
           PRINT *,  '**********  TEST_JACOBI   **********'
           DO IT  =  1,  ITMAX
!$hmpp jac region, args[A;B].noupdate=true
                DO  J  =  2, L-1
                   DO  I  =  2, L-1
                        A(I, J)  =  B(I, J)
                   ENDDO
                ENDDO
                DO  J = 2,  L-1
                   DO  I = 2,  L-1
                       B(I, J)=(A(I-1,J)+A(I,J-1)+A(I+1,J) +
          *                 A(I, J+1)) / 4
                    ENDDO
                 ENDDO
 !$hmpp jac endregion
           ENDDO
 !$hmpp jac delegatedstore, args[B]
 !$hmpp jac release
           PRINT *,B
           END
```

Алгоритм Якоби в модели PGI APM

```
      PROGRAM JACOBY_PGI_APM
      PARAMETER    (L=4096,  ITMAX=100)
      REAL     A(L,L), B(L,L)
      PRINT *,  '**********  TEST_JACOBI   **********'
!$acc data region copyin(B), copyout(B), local(A)
      DO IT  =  1,  ITMAX
!$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
      PRINT *, B
      END
```

Cray Compiling Environment 7.4.0

```
!$omp acc_region
!$omp acc_loop
          DO j = 1,M
            DO i = 2,N
                c(i,j) = a(i,j) + b(i,j)
             ENDDO
           ENDDO
!$omp end acc_loop
!$omp end acc_region
```

acc_region:

acc_copy, acc_copyin, acc_copyout, acc_shared, private, firstprivate, default(<any of above>|none), present, if(scalar-logical-expression), device(integer-expression), num_pes(depth:num [, depth:num]), async(handle)

acc_loop:

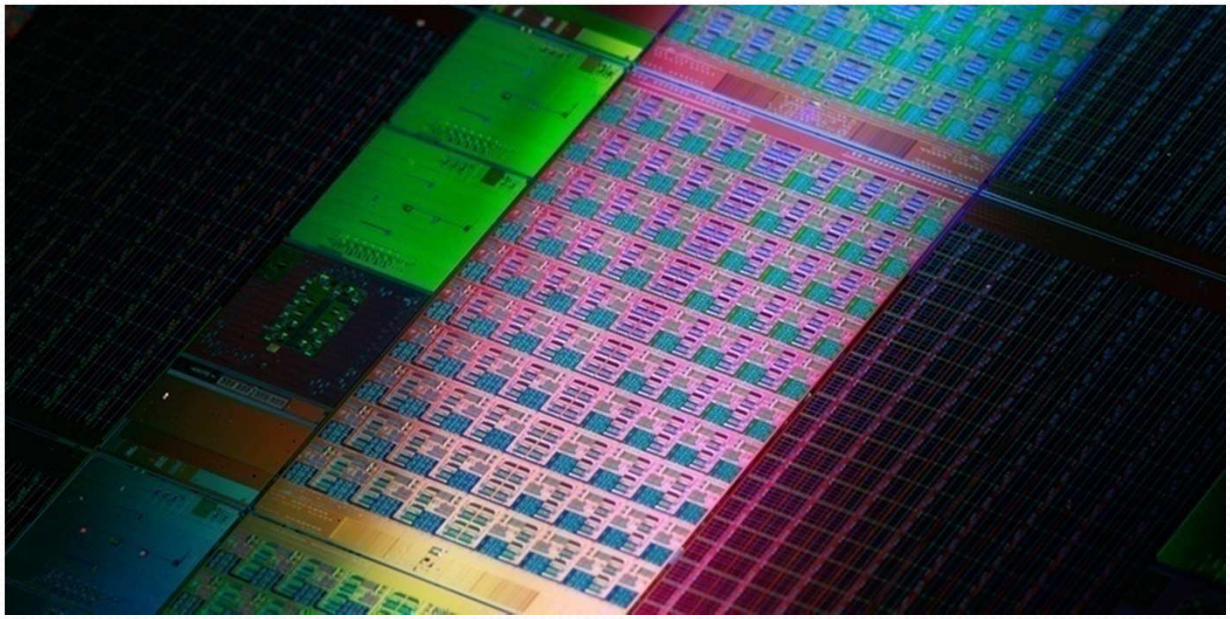
reduction(operator:list), collapse(n), schedule, cache(obj[:depth], hetero…

OpenACC

```
#pragma acc data copy(A), create(Anew)
while (iter<iter_max) {
  #pragma acc kernels loop
  for (int j = 1; j < n-1; j++) {
     for (int I = 1; I < m-1; i++) {
        Anew[j][i] = 0.25* (A[j][i+1] + A[j][i-1] +A[j-1][i] + A[j+1][i]);
     }
   }
  #pragma acc kernels loop
  for (int j = 1; j < n-1; j++) {
     for (int I = 1; i< m-1;i++ ) {
       A[j][i] = Anew[j][i];
     }
  }
  iter++;
 }
```

Intel Many Integrated Core (MIC)

```
!dir$ offload target(mic)
!$omp parallel do
        do i=1,10
            A(i) = B(i) * C(i)
        enddo
!$omp end parallel
```



OpenMP accelerator model

Новые директивы

- target
- target data
- target update
- teams
- distribute

Новые функции системы поддержки

- omp_get_num_devices
- omp_set_default_device
- omp_get_default_device
- omp_is_initial_device
- omp_get_num_teams
- omp_get_team_num

Новая переменная окружения

- OMP_DEFAULT_DEVICE

OpenMP accelerator model. Директива target

#pragma omp target *[clause[[,] clause]]*

structured-block

где clause одна из:

- **device(***integer-expression***)**
- **map** *([map-type]:list)*

map-type:

- **alloc**
- **to**
- **from**
- **tofrom (по умолчанию)**

- **if (***scalar-expression***)**

```
sum=0;
#pragma omp target device(acc0) map(A,B)
#pragma omp parallel for reduction(+: sum)
  for (i=0;i<N;i++)
      sum += A[i]*B[i];
```

OpenMP accelerator model

#pragma omp target data *[clause[[,] clause]]*

structured-block

где clause одна из:

- **device(***integer-expression***)**
- **map** *([map-type]:list)*

map-type:

- **alloc**
- **to**
- **from**
- **tofrom**

- **if (***scalar-expression***)**

#pragma omp target update*[clause[[,] clause]]*

где clause одна из:

- **to** *(list)*
- **from** *(list)*
- **device(***integer-expression***)**
- **if (***scalar-expression***)**

OpenMP accelerator model. Директива target data

```
#pragma omp target data device(acc0) map(alloc: tmp[0:N]) \
        map(to: input[:N]) map(from: output)
{
  #pragma omp target device(acc0)
  #pragma omp parallel for
    for (int i=0; i<N; i++)
        tmp[i] = some_device_computation (input[i]);

  input[0] = some_host_computation ();
  #pragma omp target update to (input[0]) device(acc0)

  #pragma omp target device(acc0)
  #pragma omp parallel for reduction(+: output)
     for (int i=0; i<N; i++) output += final_device_computation (tmp[i], input[i])
}
```

OpenMP accelerator model. Директива declare target

```
#pragma omp declare target
   function-defenition-or-declaration

#pragma omp declare target
   float Q[N][N];
   #pragma omp declare simd uniform(i) linear(j) notinbranch
   float func(const int i, const int j)
   {
      return Q[i][j] * Q[j][i];
   }
#pragma omp end declare target

…
#pragma omp target
#pragma omp parallel for reduction(+: sum)
  for (int i=0; i < N; i++) {
    for (int j=0; j < N; j++) {
        sum += func (i,j);
    }
  }

…
```

OpenMP accelerator model. Директива teams

#pragma omp teams *[clause[[,]clause] ,...]*
structured-block

где clause одна из:

- **num_teams** *(integer-expression)*
- **thread_limit** *(integer-expression)*
- **private** *(list)*
- **firstprivate** *(list)*
- **shared** *(list)*
- **default (shared | none)**
- **reduction** *(reduction-identifier: list)*

Использование директивы teams

```
float dotprod(float B[], float C[], int N)
{
  float sum0 = 0.0, sum1 = 0.0;
  #pragma omp target map(to: B[:N], C[:N])
  #pragma omp teams num_teams(2)
  {
    if (omp_get_team_num() == 0)
    {
      #pragma omp parallel for reduction(+:sum0)
      for (int i=0; i<N/2; i++)
        sum0 += B[i] * C[i];
    } else if (omp_get_team_num() == 1) {
      #pragma omp parallel for reduction(+:sum1)
      for (int i=N/2; i<N; i++)
        sum1 += B[i] * C[i];
    }
  }
  return sum0 + sum1;
}
```

OpenMP accelerator model. Директива distribute

#pragma omp distribute *[clause[[,]clause] ,...]*
for-loops

где clause одна из:

- **private** *(list)*
- **firstprivate** *(list)*
- **collapse** *(n)*
- **dist_schedule** *(kind[,: chunk_size]) // kind=static*

Может использоваться внутри конструкции **teams**.

OpenMP accelerator model. Директива distribute

```
float dotprod(float B[], float C[], int N)
{
  float sum = 0;
  int i;
  #pragma omp target teams map(to: B[0:N], C[0:N])
  #pragma omp distribute parallel for reduction(+:sum)
  for (i=0; i<N; i++)
    sum += B[i] * C[i];
  return sum;
}
```

OpenMP accelerator model. Директивы teams&&distribute

```
#pragma omp declare target
extern void func(int, int, int);

#pragma omp target device(0)
#pragma omp teams num_teams(60) num_threads (4)
// 60 physical cores, 4 threads in each team
{
  #pragma omp distribute // this loop is distributed across teams
  for (int i = 0; i < 2048; i++) {
    #pragma omp parallel for // loop is executed in parallel by 4 threads of team
    for (int j = 0; j < 512; j++) {
      #pragma omp simd  // create SIMD vectors for the machine
      for (int k=0; k<32; k++) {
          func (i,j,k);
      }
    }
  }
}
```

OpenMP accelerator model. Умножение векторов

```
extern void func_on_host();
extern void output_on_host(float *,int);
#pragma omp declare target
extern void init_on_device(float *,float *,int);

void vec_mult(float *p, int N, int dev)
{
  float *v1, *v2; int i;
  #pragma omp task shared(v1, v2) depend(out: v1, v2)
  #pragma omp target device(dev) map(v1, v2)
  {
     v1=malloc(N*sizeof(float)); v2=malloc(N*sizeof(float)); init_on_device(v1,v2,N);
  }
  func_on_host (); // execute other work asychronously
  #pragma omp task shared(v1, v2, p) depend(in: v1, v2)
  #pragma omp target device(dev) map(to: v1, v2) map(from: p[0:N])
  {
     #pragma omp parallel for
        for (i=0; i<N; i++) p[i] = v1[i] * v2[i];
     free(v1); free(v2);
  }
  #pragma omp taskwait
  output_on_host(p, N);
}
```