OpenMP 4 - What's New?

SciNet Developer Seminar

Ramses van Zon

September 25, 2013

SciNet compute • calcul CANADA

Intro to OpenMP

- For shared memory systems.
- Add parallelism to functioning serial code.
- For C, C++ and Fortran
- http://openmp.org

- Compiler/run-time does a lot of work for you
- Divides up work
- You tell it how to use variables, and what to parallelize.
- Works by adding compiler directives to code.

SciNet compute • calcul CANADA

Quick Example - C

```c
/* example1.c */
int main()
{
  int i,sum;
  sum=0;


  for (i=0; i<101; i++)
    sum+=i;

  return sum-5050;
}
```

```c
/* example1.c */
int main()
{
  int i,sum;
  sum=0;
 #pragma omp parallel
 #pragma omp for reduction(+:sum)
  for (i=0; i<101; i++)
    sum+=i;

  return sum-5050;
}
```

```
> $CC example1.c
> ./a.out
```

```
> $CC example1.c -fopenmp
> export OMP_NUM_THREADS=8
> ./a.out
```

SciNet compute • calcul CANADA

Quick Example - Fortran

```
program example1
  integer i,sum
  sum=0


  do i=1,100
     sum=sum+i
  end do

  print *, sum-5050;
end program example1
```

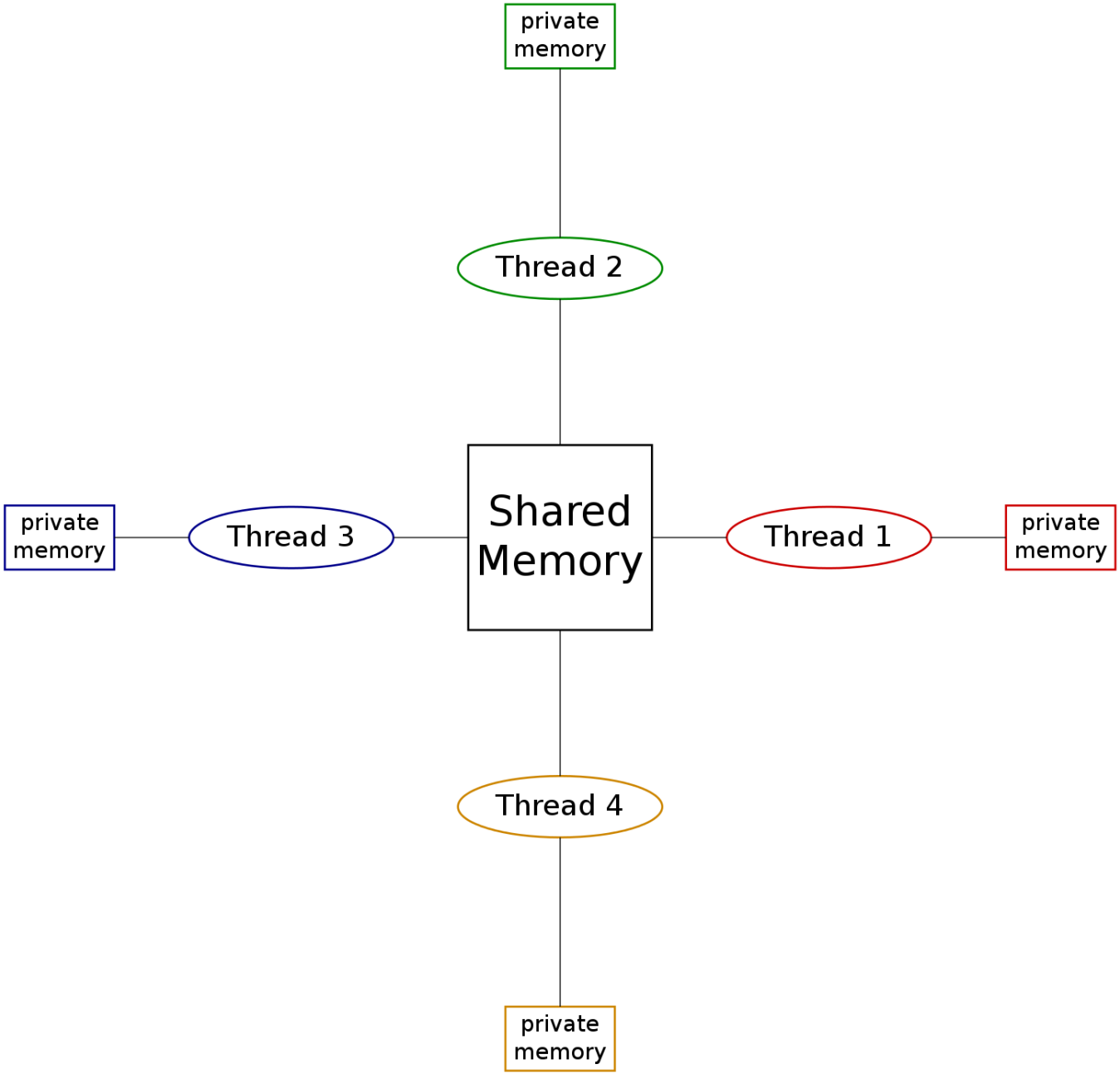
```
program example1
  integer i,sum
  sum=0
 !$omp parallel
 !$omp do reduction(+:sum)
  do i=1,100
     sum=sum+i
  end do
 !$omp end parallel
  print *, sum-5050;
end program example1
```

```
> $FC example1.f90
```

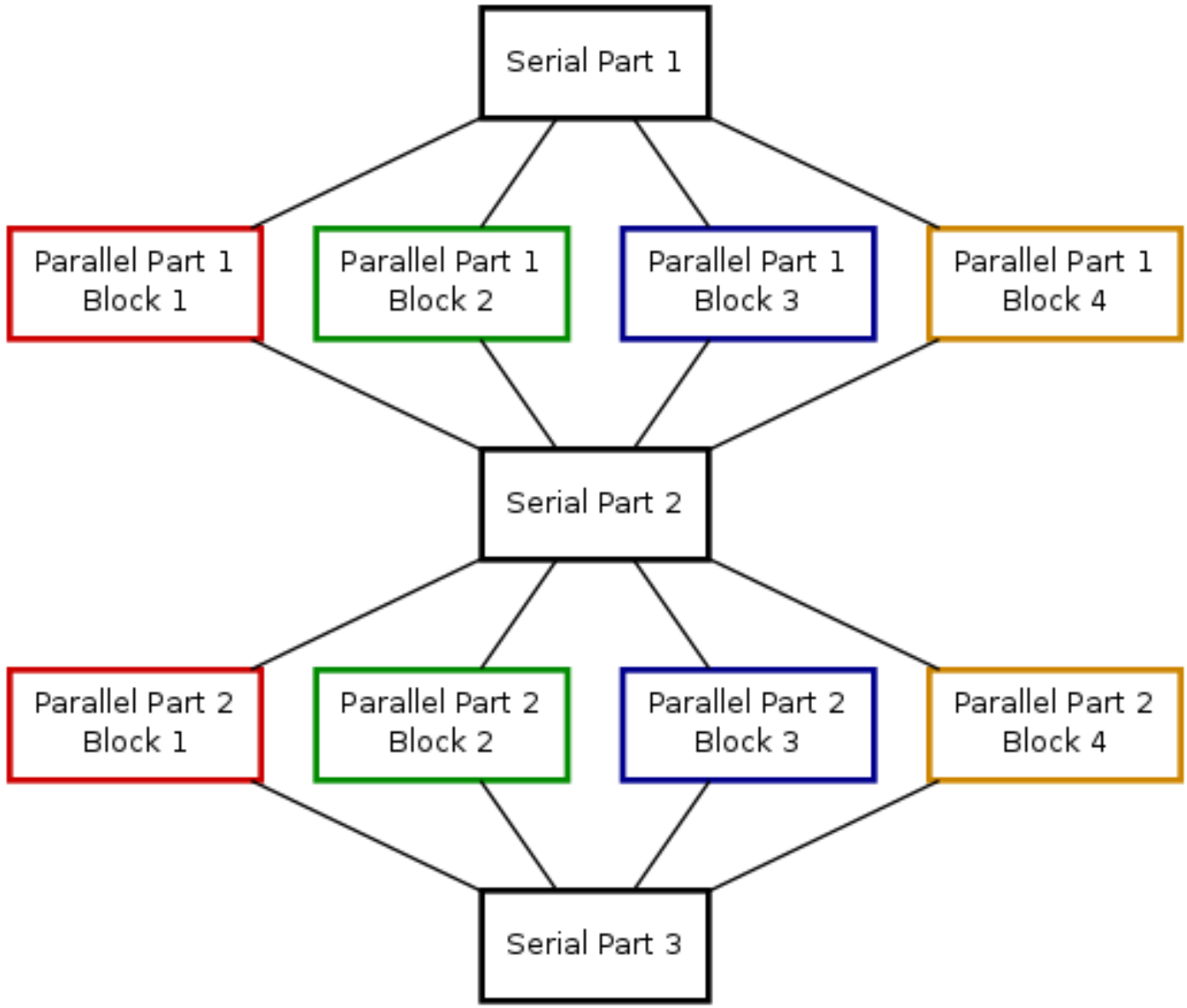
```
> $FC example1.f90 -fopenmp
```

SciNet compute • calcul CANADA

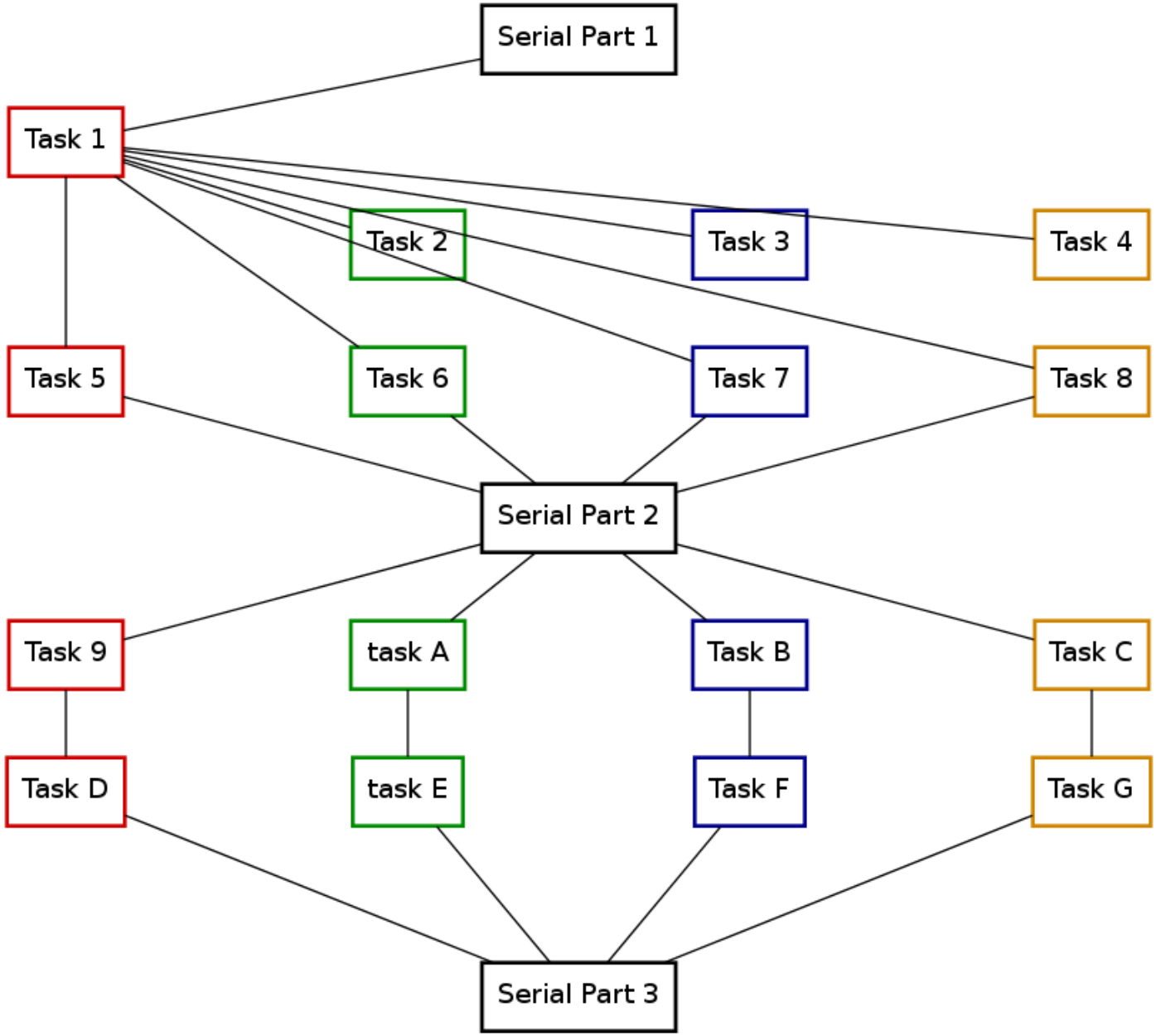
Memory Model in OpenMP (3.1)



Execution Model in OpenMP



Execution Model in OpenMP with Tasks



Existing Features (OpenMP 3.1)

1. Create threads with shared and private memory;
2. Parallel sections and loops;
3. Different work scheduling algorithms for load balancing loops;
4. Lock, critical and atomic operations to avoid race conditions;
5. Combining results from different threads;
6. Nested parallelism;
7. Generating task to be executed by threads.

Supported by GCC, Intel, PGI and IBM XL compilers.

Introducing OpenMP 4.0

- Released July 2013, OpenMP 4.0 is an API *specification*.
- As usual with standards, it's a mix of features that are commonly implemented in another form and ones that have never been implemented.
- As a result, compiler support varies. E.g. Intel compilers v. 14.0 good at offloading to phi, gcc has more task support.
- OpenMP 4.0 is 248 page document (without appendices) (OpenMP 1 C/C++ or Fortran was ≈ 40 pages)
- No examples in this specification, no summary card either.
- But it has a lot of new features...

New Features in OpenMP 4.0

1. Support for compute devices
2. SIMD constructs
3. Task enhancements
4. Thread affinity
5. Other improvements

1. Support for Compute Devices

- Effort to support a wide variety of compute devices:

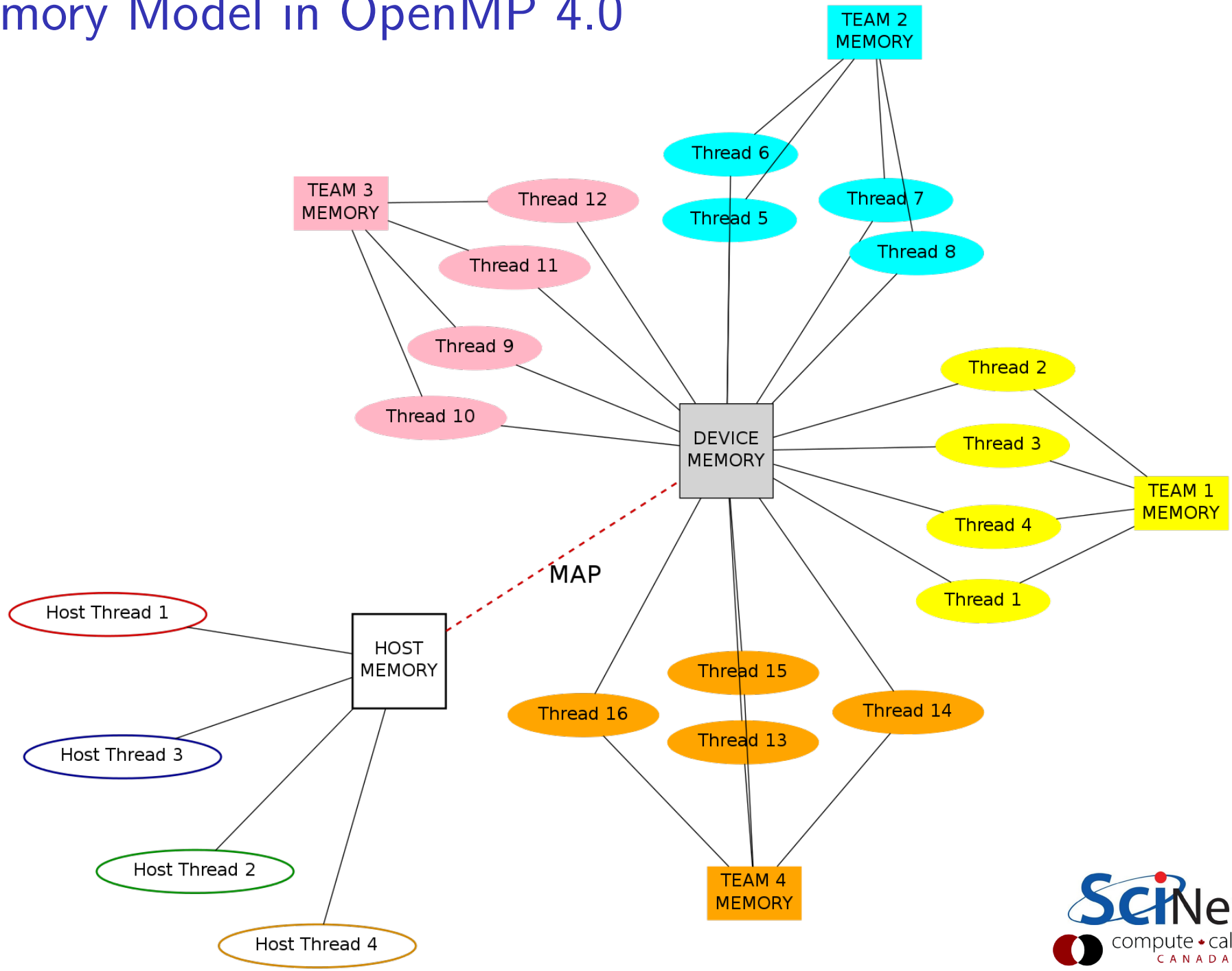
  GPUs, Xeon Phis, clusters(?)

- OpenMP 4.0 adds mechanisms to describe regions of code where data and/or computation should be moved to another computing device.

- Moves away from shared memory per se.

- `omp target`.

Memory Model in OpenMP 4.0



Memory Model in OpenMP 4.0

- Device has its own data environment
- And its own shared memory
- Threads can be bundled in a teams of threads
- These threads can have memory shared among threads of the same team
- Whether this is beneficial depends on the memory architecture of the device. (team $\approx$ CUDA thread blocks, MPI_COMM?)

Data mapping

- Host memory and device memory usually district.
- OpenMP 4.0 allows host and device memory to be shared.
- To accommodate both, the relation between variables on host and memory gets expressed as a *mapping*

  Different types:
  - `to`: existing host variables copied to a corresponding variable in the target before
  - `from`: target variables copied back to a corresponding variable in the host after
  - `tofrom`: Both `from` and `to`
  - `alloc`: Neither `from` nor `to`, but ensure the variable exists on the target but no relation to host variable.

  Note: arrays and array sections are supported.

OpenMP Device Example using target

```c
/* example2.c */
#include <stdio.h>
#include <omp.h>
int main()
{
  int host_threads, trgt_threads;
  host_threads = omp_get_max_threads();
  #pragma omp target map(from:target_threads)
  trgt_threads = omp_get_max_threads();
  printf("host_threads = %d\n", host_threads);
  printf("trgt_threads = %d\n", trgt_threads);
}
```

```
> $CC -fopenmp example2.c -o example2
> ./example2
host_threads = 16
trgt_threads = 224
```

OpenMP Device Example using target

```
program example2
  use omp_lib
  integer host_threads, trgt_threads
  host_threads = omp_get_max_threads()
  !$omp target map(from:target_threads)
  trgt_threads = omp_get_max_threads();
  !$omp end target
  print *, "host_threads =", host_threads
  print *, "trgt_threads =", trgt_threads
end program example2
```

```
> $FC -fopenmp example2.f90 -o example2
> ./example2
    host_threads = 16
    trgt_threads = 224
```

SciNet compute • calcul CANADA

OpenMP Device Example using teams, distribute

```
#include <stdio.h>
#include <omp.h>
int main()
{
   int ntprocs;
   #pragma omp target map(from:ntprocs)
   ntprocs = omp_get_num_procs();
   int ncases=2240, nteams=4, chunk=ntprocs*2;

   #pragma omp target
   #pragma omp teams num_teams(nteams) thread_limit(ntprocs/nteams)
   #pragma omp distribute
   for (int starti=0; starti<ncases; starti+=chunk)
      #pragma omp parallel for
      for (int i=starti; i<starti+chunk; i++)
         printf("case i=%d/%d by team=%d/%d thread=%d/%d\n",
                i+1, ncases,
                omp_get_team_num()+1, omp_get_num_teams(),
                omp_get_thread_num()+1, omp_get_num_threads());
}
```

OpenMP Device Example using teams, distribute

```
program example3
 use omp_lib
 integer i, ntprocs, ncases, nteams, chunk
 !$omp target map(from:ntprocs)
 ntprocs = omp_get_num_procs()
 !$omp end target
 ncases=2240
 nteams=4
 chunk=ntprocs*2
 !$omp target
 !$omp teams num_teams(nteams) thread_limit(ntprocs/nteams)
 !$omp distribute
 do starti=0,ncases,chunk
  !$omp parallel do
  do i=starti,starti+chunk
   print *,"i=",i,"team=",omp_get_team_num(),"thread=",omp_get_thread_num()
  end do
  !$omp end parallel
 end do
 !$omp end target
end program example3
```

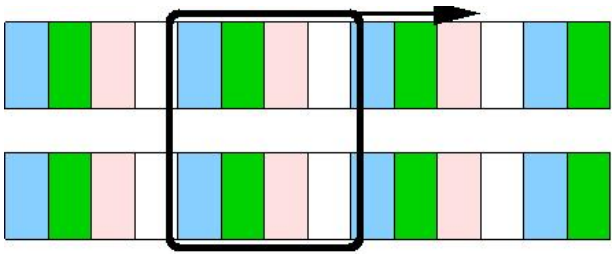
SciNet compute • calcul CANADA

Summary of New Directives and Functions for Devices

- `omp target [map]`
  marks a region to execute on device
- `omp teams`
  creates a league of thread teams
- `omp distribute`
  distributes a loop over the teams in the league
- `omp declare target` / `omp end declare target`
  marks function(s) that can be called on the device

- `omp_get_team_num()`
- `omp_get_team_size()`
- `omp_get_num_devices()`

2. SIMD Constructs



- OpenMP can enable vectorization of both serial as well as parallelized loops.
- *vectorization* = processing multiple elements of an array at the same time.
- This is done using SIMD instructions.
- SIMD=single instruction multiple data. Usually 2, 4,or 8 *SIMD lanes* wide.
- Can also indicate to OpenMP to create versions of functions that can be invoked across SIMD lanes.

New Directives for SIMD Support

- `omp simd`
  marks a loop to be executed using SIMD lanes
- `omp declare simd`
  marks a function that can be called from a SIMD loop
- `omp parallel for simd`
  marks a loop for thread work-sharing as well as SIMDing

OpenMP SIMD Loop Example

```
#include <stdio.h>
#define N 262144
int main()
{
   long long d1=0;
   double a[N], b[N], c[N], d2=0.0;
   #pragma omp simd reduction(+:d1)
   for (int i=0;i<N;i++)
      d1+=i*(N+1-i);
   #pragma omp simd
   for (int i=0; i<N;i++) {
      a[i]=i;
      b[i]=N+1-i;
   }
   #pragma omp parallel for simd reduction(+:d2)
   for (int i=0; i<N; i++)
      d2+=a[i]*b[i];
   printf("result1 = %ld\nresult2 = %.2lf\n", d1, d2);
}
```

OpenMP SIMD Loop Example

```
program simdex
  integer, parameter :: N = 262144
  integer(kind=8) :: i, d1
  real(kind=8), dimension(N) :: a, b, c
  real(kind=8) :: d2
  d1=0 ; d2=0.
  !$omp simd reduction(+:d1)
  do i=1,N
     d1 = d1 + (i-1)*(N-i)
  end do
  !$omp end simd
  !$omp simd
  do i=1,N
     a(i)=i-1 ; b(i)=N-i
  end do
  !$omp end simd
  !$omp parallel do simd reduction(+:d2)
  do i=1,N
     d2 = d2 + a(i)*b(i)
  enddo
  !$omp end parallel
  print *,"result1 =",d1,"result2 =",d2
end program simdex
```

SciNet compute • calcul CANADA

OpenMP SIMD Function Example

```
#include <stdio.h>
#pragma omp declare simd
double computeb(int i)
{ return N+1-i; }
#define N 262144
int main()
{
   long long d1=0;
   double a[N], b[N], c[N], d2=0.0;
   #pragma omp simd reduction(+:d1)
   for (int i=0;i<N;i++)
      d1 += i*computeb(i);
   #pragma omp simd
   for (int i=0; i<N;i++) {
      a[i]=i; b[i]=computeb(i);
   }
   #pragma omp parallel for simd reduction(+:d2)
   for (int i=0; i<N; i++)
      d2 += a[i]*b[i];
   printf("result1 = %ld\nresult2 = %.2lf\n", d1, d2);
}
```

SciNet compute • calcul CANADA

3. Task Enhancements

- Can abort parallel OpenMP execution by conditional cancellation at implicit and user-defined cancellation points.
- Tasks can be grouped to into task groups can be aborted to reflect completion of cooperative tasking activities such as search.
- Task-to-task synchronization is supported through the specification of task dependency.

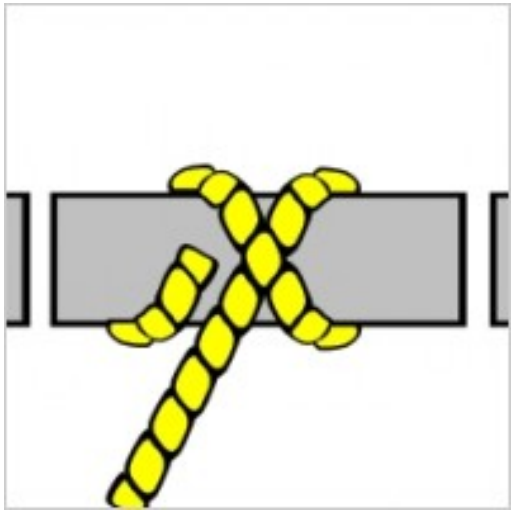
OpenMP Task Cancellation Example

```
#include <stdio.h>
#define N 40
int main()
{
   char haystack[N+1]="abcabcabczabcabcabcxabcabcabczabcabcabcz";
   char needle='x';
   int pos;
   #pragma omp parallel for
   for (int i=0; i<N; i++) {
      if (haystack[i]==needle) {
         pos=i;
         #ifndef _OPENMP
         break;
         #else
         #pragma omp cancel for
         #endif
      }
   }
   printf("\n'%c' found at position %d in %s\n",needle,pos,haystack);
}
```

Overview of New Directives and Functions for Tasks

- `omp cancel parallel|for|sections|taskgroup`
  starts cancellation of all tasks in the same construct
- `omp cancelation point parallel|for|sections|taskgroup`
  marks a point at which this task may be canceled
- `omp taskgroup`
  marks a region such that all tasks started in it belong to a group
- `omp task depend([in|out|inout]:variable)` clause
  marks that a task depends on other task

4. Thread Affinity



- OpenMP can now be told better where to execute threads.
- Can be used to get better locality, less false sharing, more memory bandwidth.
- To specify platform-specific data: Environment variable `OMP_PLACES`
- To describe thread binding to processor:
  - Environment variable: `OMP_PROC_BIND`
  - In code using `omp parallel`'s new `proc_bind` clause.

Allowed values:
`false, true, master, close, spread`

Example of Specifying Affinity

```
> $CC example.c -fopenmp -o example

> export OMP_NUM_THREADS=16

> export OMP_PLACES=0,8,1,9,2,10,3,11,4,12,5,13,6,14,7,15

> export OMP_PROC_BIND=spread,close

> ./example

...
```

5. Other improvements

- User-defined reductions:
  Previously, OpenMP API only supported reductions with base language operators and intrinsic procedures. With OpenMP 4.0 API, user-defined reductions are now also supported.

  ```
  omp declare reduction
  ```

- Sequentially consistent atomics:
  A clause has been added to allow a programmer to enforce sequential consistency when a specific storage location is accessed atomically.

  ```
  omp atomic seq_cst
  ```

- Optional dump all internal variables at program start:

  ```
  OMP_DISPLAY_ENV=TRUE|FALSE|VERBOSE
  ```

SciNet compute • calcul CANADA

Thank you for your attention.

Have fun exploring!

http://openmp.org/wp/openmp-specifications

SciNet
compute • calcul
CANADA