Introduction to OpenMP

Dilip Angom

angom@prl.ernet.in

Theoretical Physics and Complex Systems

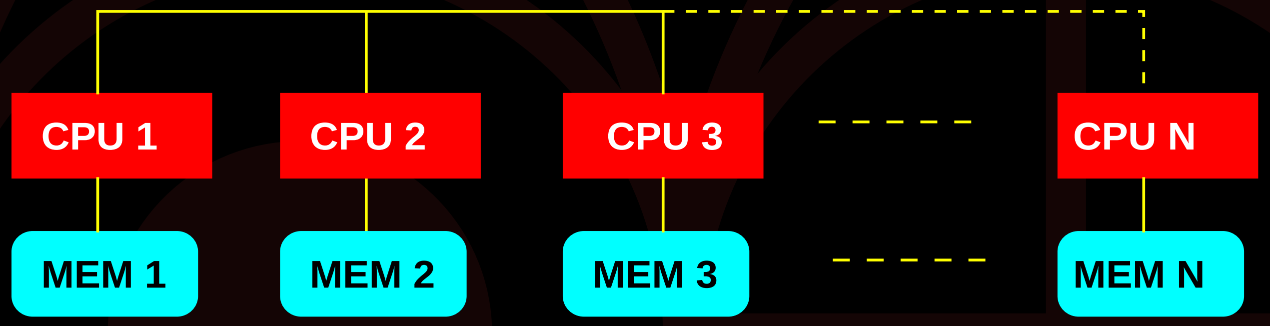
Physical Research Laboratory

Why OpenMP

There are limitations in message passing libraries based parallel programming (distributed memory systems).

Why OpenMP

There are limitations in message passing libraries based parallel programming (distributed memory systems).



Why OpenMP

There are limitations in message passing libraries based parallel programming (distributed memory systems).

- User synchronizes the processors explicitly.
- Debugging is non trivial as there is no common image.
- Data duplication across the processors.

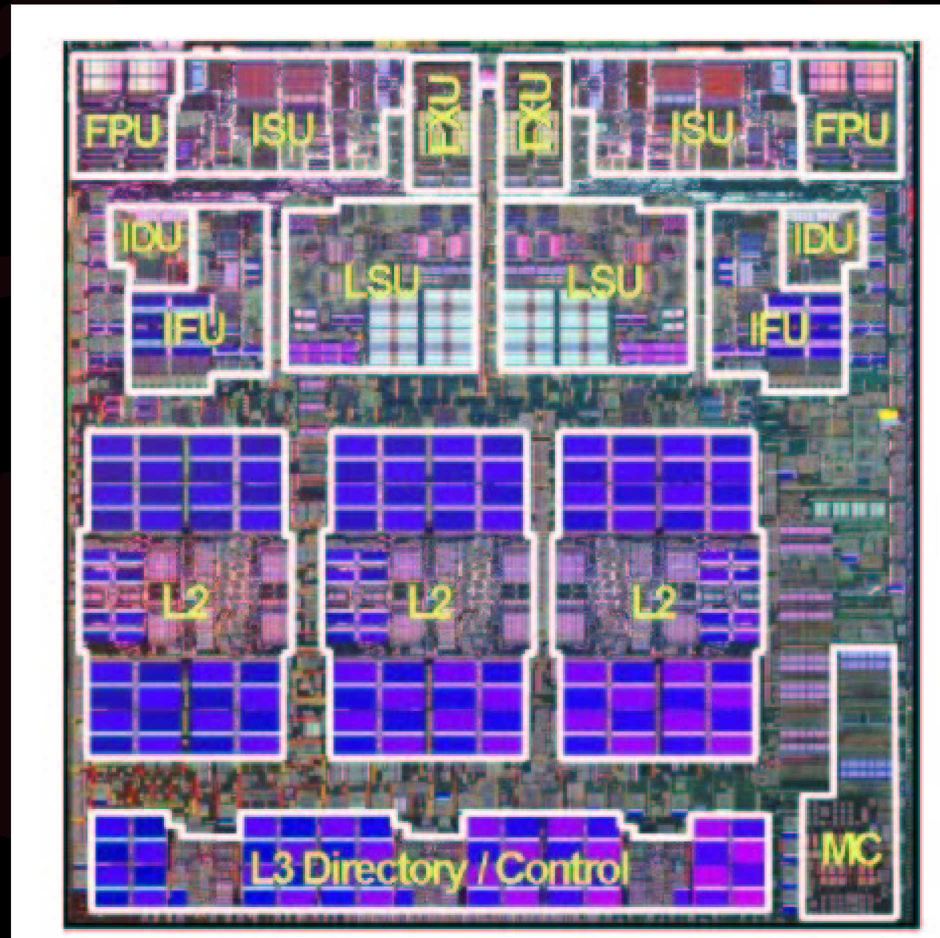
Why OpenMP

There are limitations in message passing libraries based parallel programming (distributed memory systems).

- User synchronizes the processors explicitly.
- Debugging is non trivial as there is no common image.
- Data duplication across the processors.

Lack of an industry wide standard in shared memory high performance computing. The microprocessors are based on the SMP architecture.

POWER5 (IBM) Processor Chip



FXU - Fixed Point (Integer) Unit
FPU - Floating Point Unit
ISU - Instruction Sequencing Unit
IDU - Instruction Decoding Un
LSU - Load Store Unit
IFU - Instruction Fetch Unit
L2 - Level 2 Cache
L3 - Level 3 Cache
MC - Memory Controller

Why OpenMP

There are limitations in message passing libraries based parallel programming (distributed memory systems).

- User synchronizes the processors explicitly.
- Debugging is non trivial as there is no common image.
- Data duplication across the processors.

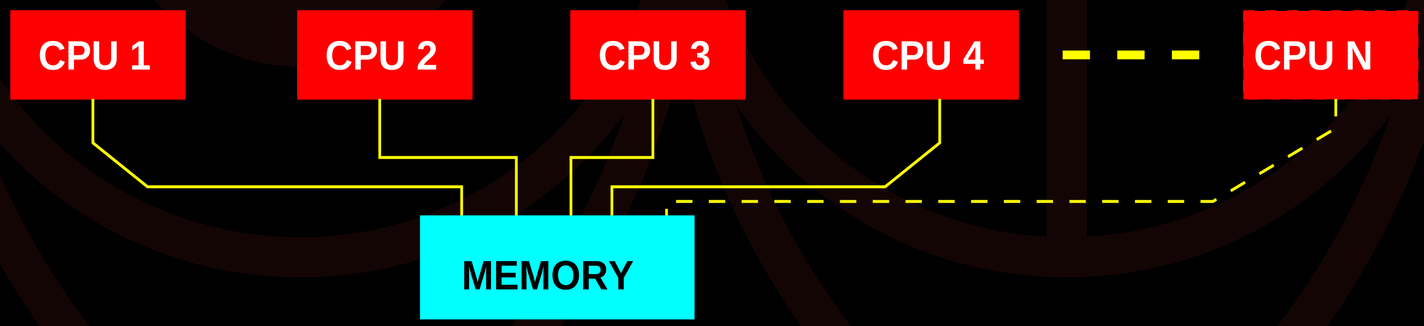
Lack of an industry wide standard in shared memory high performance computing. The microprocessors are based on the SMP architecture.

Why OpenMP

There are limitations in message passing libraries based parallel programming (distributed memory systems).

- User synchronizes the processors explicitly.
- Debugging is non trivial as there is no common image.
- Data duplication across the processors.

Lack of an industry wide standard in shared memory high performance computing. The microprocessors are based on the SMP architecture.



Why OpenMP

There are limitations in message passing libraries based parallel programming (distributed memory systems).

- User synchronizes the processors explicitly.
- Debugging is non trivial as there is no common image.
- Data duplication across the processors.

Lack of an industry wide standard in shared memory high performance computing. The microprocessors are based on the SMP architecture.

- POWER5 (eSeries Servers) IBM (http://www.redbooks.ibm.com/redpieces/pdfs/sg245768.pdf)
- UltraSPARC IV (Sun Fire Servers) Sun Microsystems (http://www.sun.com/processors/whitepapers/us4_whitepaper.pdf)
- PA-8800 (HP 9000 Servers) Hewlett Packard

What is OpenMP

It is an Application Program Interface (API) to provide a model for parallel programming portable across different shared memory architectures.

What is OpenMP

It is an Application Program Interface (API) to provide a model for parallel programming portable across different shared memory architectures.

What is an API?

Application Program Interface

A set of routines, protocols, and tools for building software applications. A good API makes it easier to develop a program by providing all the building blocks. A programmer puts the blocks together.

Application Program Interface

A set of routines, protocols, and tools for building software applications. A good API makes it easier to develop a program by providing all the building blocks. A programmer puts the blocks together.

APIs compromises flexibility, however provides standard and higher level of abstraction to programmers. Consider the component tasks to print hello world on the screen.

1. Outline the shapes of the letters H, e, l, l, o, W, o, r, l, d.
2. Locate a matrix of black and white squares resembling these letters.
3. Program the CPU to put this matrix into the display adapter's frame buffer.
4. Set the graphics card to scan its frame buffer and generate the signal.

Application Program Interface

A set of routines, protocols, and tools for building software applications. A good API makes it easier to develop a program by providing all the building blocks. A programmer puts the blocks together.

APIs compromises flexibility, however provides standard and higher level of abstraction to programmers. Consider the component tasks to print hello world on the screen.

1. Outline the shapes of the letters H, e, l, l, o, W, o, r, l, d.
2. Locate a matrix of black and white squares resembling these letters.
3. Program the CPU to put this matrix into the display adapter's frame buffer.
4. Set the graphics card to scan its frame buffer and generate the signal.

A much simpler option is to use an API

1. Write an HTML document containing hello world.
2. Open the document using a web browser.

What is OpenMP

It is an Application Program Interface (API) to provide a model for parallel programming portable across different shared memory architectures and scalable.

Standard is jointly defined by a group with members from major computer hardware and software vendors like IBM, Silicon Graphics, Hewlett Packard, Intel, Sun Microsystems, The Portland Group, etc.

What is OpenMP

It is an Application Program Interface (API) to provide a model for parallel programming portable across different shared memory architectures and scalable.

Standard is jointly defined by a group with members from major computer hardware and software vendors like IBM, Silicon Graphics, Hewlett Packard, Intel, Sun Microsystems, The Portland Group, etc.

OpenMP API consists of the following components:

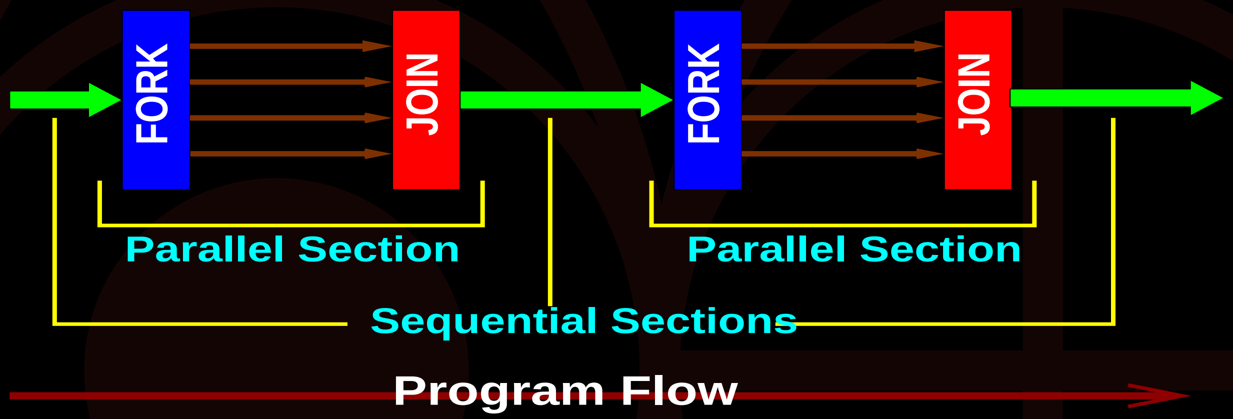
- Compiler directives
  Instructs the compiler to process the code section following the directive for parallel execution.
- Library routines
  Routines that affect and monitor threads, processors and environment variables. It also has routines to control thread synchronization and get timings.
- Environment variables
  Variables controlling the execution of the OpenMP program.

Execution and Memory Models

Parallel execution in OpenMP is based on the fork-join model, where the master thread creates a team of threads for parallel execution.

Execution and Memory Models

Parallel execution in OpenMP is based on the fork-join model, where the master thread creates a team of threads for parallel execution.



Execution and Memory Models

Parallel execution in OpenMP is based on the fork-join model, where the master thread creates a team of threads for parallel execution.

- Program execution begins as a single thread of execution, called the initial thread.
- A thread encountering a `parallel` construct becomes a master, creates a team of itself and additional threads.
- All members of the team execute the code inside `parallel` construct.

Execution and Memory Models

Parallel execution in OpenMP is based on the fork-join model, where the master thread creates a team of threads for parallel execution.

- Program execution begins as a single thread of execution, called the initial thread.
- A thread encountering a `parallel` construct becomes a master, creates a team of itself and additional threads.
- All members of the team execute the code inside `parallel` construct.

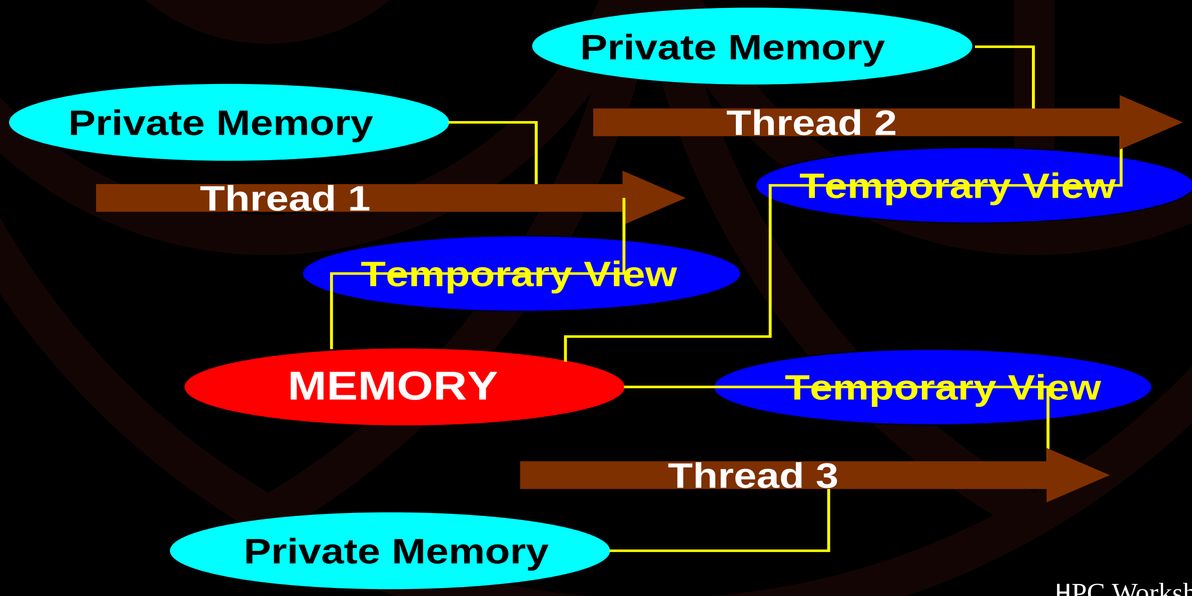
Each thread has a temporary view of the memory, which is like a cache, and a private memory not accessible other threads.

Execution and Memory Models

Parallel execution in OpenMP is based on the fork-join model, where the master thread creates a team of threads for parallel execution.

- Program execution begins as a single thread of execution, called the initial thread.
- A thread encountering a `parallel` construct becomes a master, creates a team of itself and additional threads.
- All members of the team execute the code inside `parallel` construct.

Each thread has a temporary view of the memory, which is like a cache, and a private memory not accessible other threads.



Execution and Memory Models

Parallel execution in OpenMP is based on the fork-join model, where the master thread creates a team of threads for parallel execution.

- Program execution begins as a single thread of execution, called the initial thread.
- A thread encountering a `parallel` construct becomes a master, creates a team of itself and additional threads.
- All members of the team execute the code inside `parallel` construct.

Each thread has a temporary view of the memory, which is like a cache, and a private memory not accessible other threads.

- Relaxed consistency, the thread's view of memory is not required to be consistent with the memory at all times.
- Flush operation causes the last modified variable in the temporary view to be written to memory.
- Private variables of a thread can be copies of data from memory and cannot be accessed by other threads.

The `parallel` construct can be nested arbitrary number of times. Thread encountering `parallel` becomes the master.

How OpenMP Works

In FORTRAN comment character followed by a sentinel indicates the line of code is OpenMP directive.

```
!$omp OpenMP directive
c$omp OpenMP directive
```

How OpenMP Works

In FORTRAN comment character followed by a sentinel indicates the line of code is OpenMP directive.

`parallel` is the OpenMP directive which creates a team of threads.

```
!$omp OpenMP directive
c$omp OpenMP directive

!$omp parallel
!$omp parallel
```

How OpenMP Works

In FORTRAN comment character followed by a sentinel indicates the line of code is OpenMP directive.

parallel is the OpenMP directive which creates a team of threads.

Work is distributed among the threads using work-sharing OpenMP directives `loop`, `section` and `single`.

```
!$omp OpenMP directive
c$omp OpenMP directive

!$omp parallel
c$omp parallel

!$omp parallel
!$omp do/section/single
```

How OpenMP Works

In FORTRAN comment character followed by a sentinel indicates the line of code is OpenMP directive.

`parallel` is the directive in OpenMP which creates a team of threads.

Work is distributed among the threads using work-sharing OpenMP directives `loop`, `section` and `single`.

Each of these directives have associated clauses to control data sharing and mode of execution.

```
!$omp OpenMP directive
c$omp OpenMP directive

!$omp parallel
c$omp parallel

!$omp parallel
!$omp do/section/single

!$omp parallel[clause]
!$omp workshare[clause]
```

How OpenMP Works

In FORTRAN comment character followed by a sentinel indicates the line of code is OpenMP directive.

parallel is the directive in OpenMP which creates a team of threads.

Work is distributed among the threads using work-sharing OpenMP directives `loop`, `section` and `single`.

Each of these directives have associated clauses to control data sharing and mode of execution.

end directive terminates the work share and parallel constructs.

```
!$omp OpenMP directive
c$omp OpenMP directive

!$omp parallel
c$omp parallel

!$omp parallel
!$omp do/section/single

!$omp parallel[clause]
!$omp workshare[clause]

!$omp end workshare
!$omp end parallel
```

Parallel Execution in OpenMP

```
........................
sequential block
```

Parallel Execution in OpenMP

```
      ........................
      sequential block
!$omp parallel [clause[[,]clause]...]
      ........................
where clause is one of the following
      if(scalar-logical-expression)
      private(list)
      firstprivate(list)
      default(private|shared|none )
      shared(list)
      copyin(list)
      reduction(...)
      num_threads(...)
```

Parallel Execution in OpenMP

```
      ........................
      sequential block
!$omp parallel [clause[[,]clause]...]
      where clause can be if, private,
      firstprivate, default, shared,
      copyin, reduction, num_threads
!$omp do|section|single[clause[...]...]
where clause is one of the following
      private(list)
      firstprivate(list)
      lastprivate(list)
      reduction(...)
      ordered
      schedule(kind[,chunk_size])
```

Parallel Execution in OpenMP

```
      .........................
      sequential block
!$omp parallel [clause[[,]clause]...]
      where clause can be if, private,
      firstprivate, default, shared,
      copyin, reduction, num_threads
!$omp do|section|single[clause[...]...]
      where clause can be private,
      firstprivate, lastprivate,
      reduction, ordered, schedule
      do ia = 1, N
        structured block
      end do
      [!$omp section]
        structured block
      [!$omp section]
        structured block
      ...
```

Parallel Execution in OpenMP

```
      ........................
      sequential block
!$omp parallel [clause[[,]clause]...]
      where clause can be if, private,
      firstprivate, default, shared,
      copyin, reduction, num_threads
!$omp do|section|single[clause[...]...]
      where clause can be private,
      firstprivate, lastprivate,
      reduction, ordered, schedule
      do ia = 1, N
        structured block
      end do
      [!$omp section]
        structured block
      [!$omp section]
        structured block
      ...
!$omp end do|section[nowait]||
!$&   single[end_clause...]
!$omp end parallel
      sequential block
      ........................
```

Workshare Directives

Workshare directives follows after a `parallel` directive or it can occur in a region where there are multiple threads.

There are three possible workshare constructs.

- loop
  It distributes the work load of the do which immediately follows it.
- sections
  A set of sections follows the directive and these are executed by a team of threads.
- single
  The section of code enclosed within this workshare directive is executed by a single thread.

The generic format of the workshare directives is

```
!$omp workshare [clause[[,]clause]...]
      ..........
      lines of code
      ..........
!$omp end workshare
```

where `workshare` can be one of the three directives.

Data Scope Attribute Clauses

Data scope attribute clauses allow specifying scope of data to each thread. There are five important data scope attributes.

- private
  Each thread in the team has its own uninitialized local copy of the variables and common blocks listed as `private`. A variable should be defined as `private`, if it's value is not dependent on any other thread.

- firstprivate
  Each thread has its own initialized local copy of the variables and common blocks listed as `firstprivate`.

- lastprivate
  Variables or common blocks listed as `lastprivate` can be referred to outside of the construct of the directive. The assigned value is the last calculated in the construct.

Data Scope Attribute Clauses (contd)

- copyin

  Variables or common blocks listed in the `copyin` clause are duplicated privately for each thread from the master thread's copy.

- shared

  Variables and common blocks declared as `shared` are available to all the threads.

Synchronization Directives

- critical

  The blocks of code enclosed within `critical` construct are executed by one thread at a time.

- barrier

  This directive synchronizes all the threads. When a thread encounters the `barrier` directive, it will wait untill all other threads in the team reach the same point.

- atomic

  The `atomic` ensures that only one thread is writing to a specific memory location. The directive binds to the executable statement which follows immediately.

Synchronization Directives (contd)

- flush

  The `flush` directive ensures that each thread has access to data generated by other threads. It makes the memory of the threads consistent.

- ordered

  This directive cause the iteration of a block of code within a parallel loop to be executed as it would do sequentially.

`parallel` and `do` directives

Consider the multiplication of two square matrices A and B to get the matrix C.

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

```
      do ii = 1, nmax
        do jj = 1, nmax
          sum = 0.

!         +------------+
!         | Inner Loop |
!         +------------+

          do kk = 1, nmax
            sum = sum + A(ii,kk)*B(kk,jj)
          enddo
          C(ii,jj) = sum
        enddo
      enddo
```

`parallel` and `do` directives

```
!$omp parallel
!$omp do
      do ii = 1, nmax
        do jj = 1, nmax
          sum = 0.

!         +------------+
!         | Inner Loop |
!         +------------+

          do kk = 1, nmax
            sum = sum + A(ii,kk)*B(kk,jj)
          enddo
          C(ii,jj) = sum
        enddo
      enddo
!$omp end do
!$omp end parallel
```

What about the data attributes?

`parallel` and `do` directives

```
!$omp parallel private (ii, jj, kk, sum)
!$omp do
     do ii = 1, nmax
       do jj = 1, nmax
         sum = 0.

!        +------------+
!        | Inner Loop |
!        +------------+
         do kk = 1, nmax
           sum = sum + A(ii,kk)*B(kk,jj)
         enddo
         C(ii,jj) = sum
       enddo
     enddo
!$omp end do
!$omp end parallel
```

`parallel` and `sections` directives

Consider the multiplication of two square matrices again.

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

```
      do ii = 1, nmax
         do jj = 1, nmax
            sum = 0.

!           +------------+
!           | Inner Loop |
!           +------------+

            do kk = 1, nmax
               sum = sum + A(ii,kk)*B(kk,jj)
            enddo
            C(ii,jj) = sum
         enddo
      enddo
```

parallel and sections directives

```
      do ii = 1, nmax
!$omp parallel private (ii, jj, kk, sum)
!$omp sections
!$omp section
        do jj = 1, nmax/2
          sum = 0.
          do kk = 1, nmax
            sum = sum + A(ii,kk)*B(kk,jj)
          enddo
          C(ii,jj) = sum
        enddo
!$omp section
        do jj = 1 + nmax/2, nmax
          sum = 0.
          do kk = 1, nmax
            sum = sum + A(ii,kk)*B(kk,jj)
          enddo
          C(ii,jj) = sum
        enddo
!$omp end sections
!$omp end parallel
      end do
```

Library Functions

In addition to the OpenMP `directives`, there are library to get and set the execution environment control variables.

- omp_get_thread_num ()
  The function returns the number (integer) of the currently executing thread within the team.

- omp_get_max_threads ( )
  It returns the maximum number (integer) of threads that can execute concurrently in a single `parallel` section.

- omp_get_num_procs ( )
  It returns the current number (integer) of online processors on the machine.

Library Functions (contd)

- omp_get_dynamic( ) and omp_set_dynamic( )
  The first function returns .TRUE (logical), if dynamic thread adjustment is enabled. The second function can set the status of the dyanmic thread adjustment.

- omp_get_num_threads and omp_set_num_threads
  The first function returns the number (integer) of threads in the current `parallel` section and the second function sets the number of threads to execute the parallel section.

- omp_get_nested and omp_set_nested
  The first function returns .TRUE. (logical), if nested parallelism is enabled. The second function can enable or disable nested parallelism.

Bibliography

1. POWER5 (eSeries Servers) IBM
   http://www.redbooks.ibm.com/redpieces/pdfs/sg248000.pdf
   http://www.redbooks.ibm.com/redpieces/pdfs/sg245768.pdf
2. UltraSPARC IV (Sun Fire Servers) Sun Microsystems
   http://www.sun.com/processors/feature/USFamilyBrochure_FINAL.pdf
   http://www.sun.com/processors/whitepapers/USIIICuoverview.pdf
   http://www.sun.com/processors/whitepapers/us4_whitepaper.pdf
3. OpenMP
   http://www.openmp.org/drupal/mp-documents/draft_spec25.pdf