C-DAC Four Days Technology Workshop

ON

Hybrid Computing – Coprocessors/Accelerators
Power-Aware Computing – Performance of Applications Kernels

hyPACK-2013
Mode 3 : OpenMP 4.0

Lecture Topic :
OpenMP 4.0- An Overview

***Venue** : CMSD, UoHYD ; **Date** : October 15-18, 2013*

An Overview of OpenMP 4.0

Lecture Outline

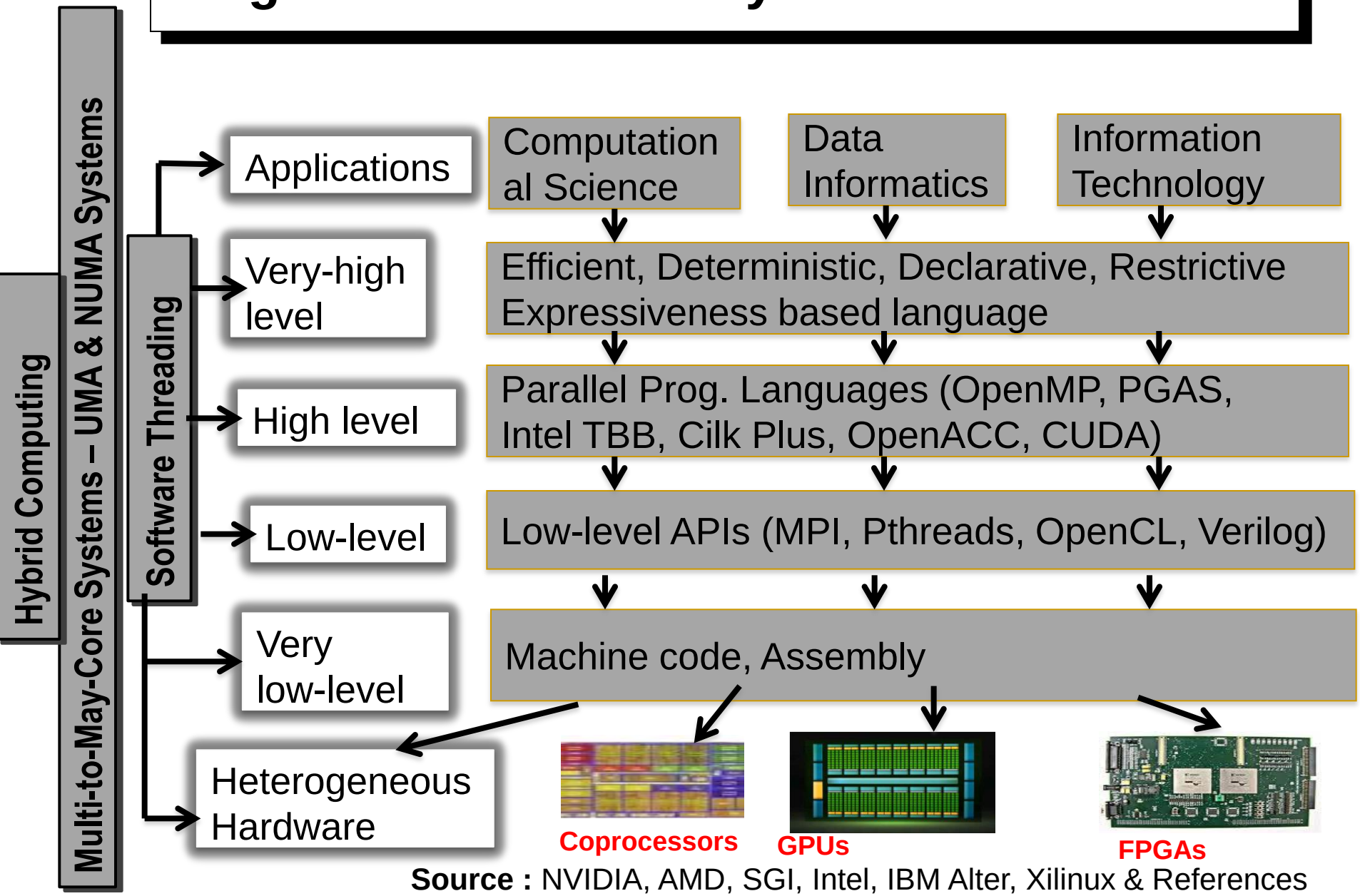
Following topics will be discussed

- Understanding of Xeon-Phi Architecture- OpenMP
- Programming on Devices using “Accelerator Prog.” & OpenMP 4.0
- Tuning & Performance – Software Threading-OpenMP 4.0

Programming on Systems with Co-processors - OpenMP 4.0

Systems with Devices - Overview

Prog.API - Multi-Core Systems with Devices



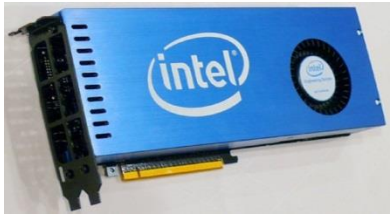
Source : NVIDIA, AMD, SGI, Intel, IBM Alter, Xilinux & References

Multi-Core Systems with Accelerator Types

- **FPGA**
  - Xilinx, Alter

- **GPU**
  - Nvidia (Kepler),
  - AMD Trinity APU

- **MIC (Intel Xeon-Phi)**
  - Intel Xeon-Phi (MIC)



Source : NVIDIA, AMD, SGI, Intel, IBM Alter, Xilinux References

Prog.API - Multi-Core Systems with Devices

DO Parallel
DO Synchronize
Get Maximum of all values
Give to Compiler

DO TRANSFER from Host to Device
Perform C omp. On Device

Use as Linux OS
DO TRANSFER from Host to Device

Parallel Prog. Languages (OpenMP, PGAS, Intel TBB, Cilk Plus, OpenACC, CUDA)

Programming with High Level APIs

Host : Multi-Core Systems OR ARM Multi-core Systems

CPU- Multi-Core Sys With

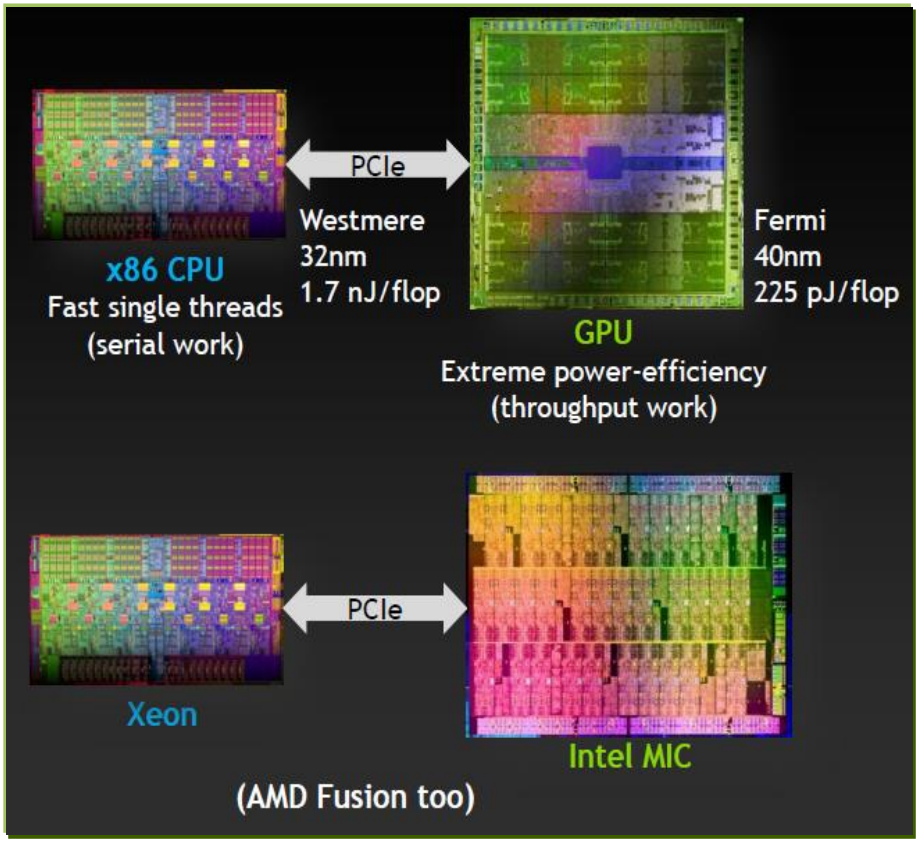
FPGA (Xilinx, Alter)

GPU - Nvidia (Kepler), AMD Trinity APU

MIC (Intel Xeon-Phi)

Source : NVIDIA, AMD, SGI, Intel, IBM Alter, Xilinux References

HPC Going Hybrid



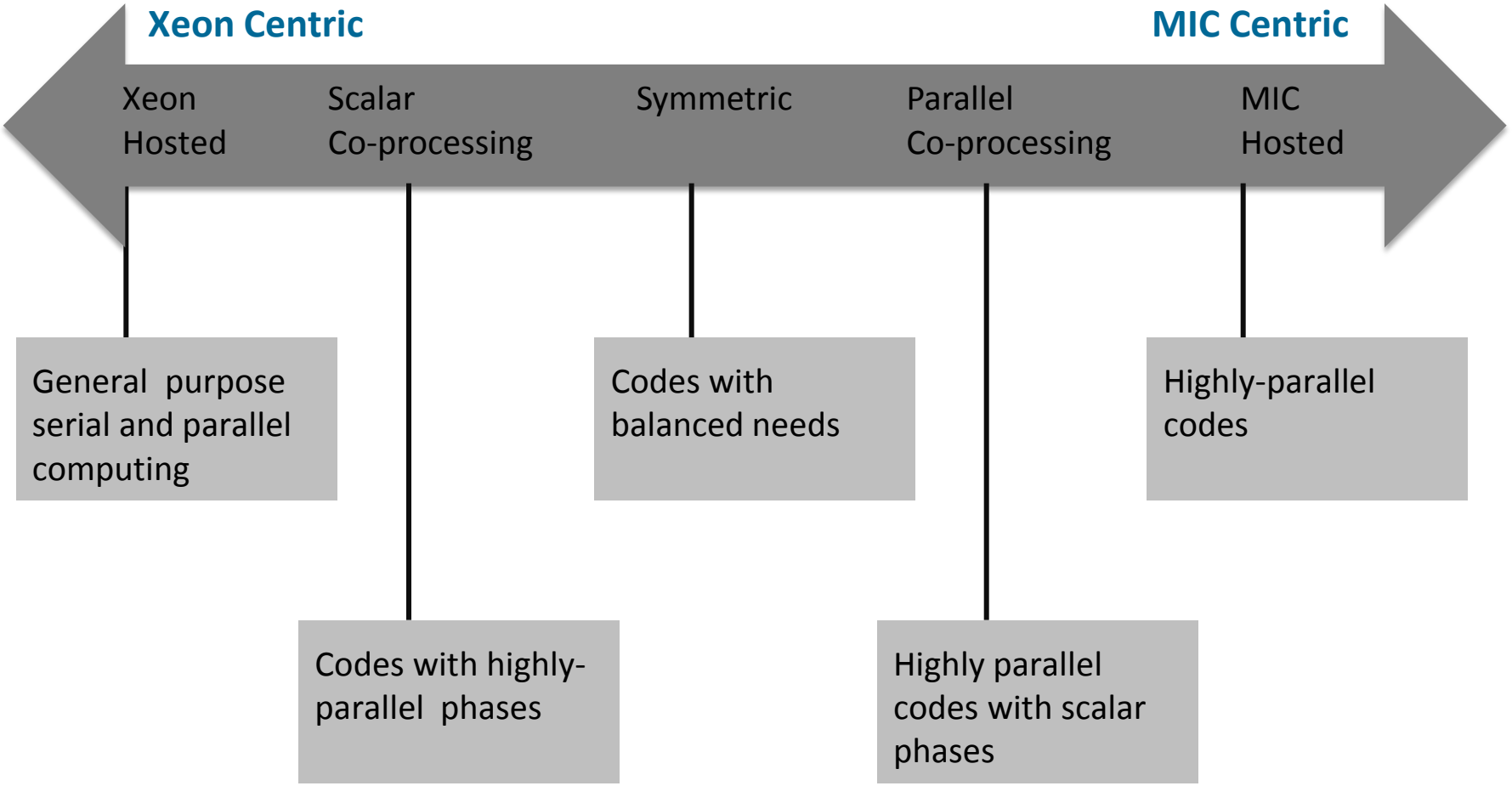
- Do most work by cores optimized for extreme energy efficiency
- Still need a few cores optimized for fast serial work

- And memory hierarchies are getting deeper…
  - More levels
  - Large impact on energy usage

Source : NVIDIA, PGI & References given in the presentation

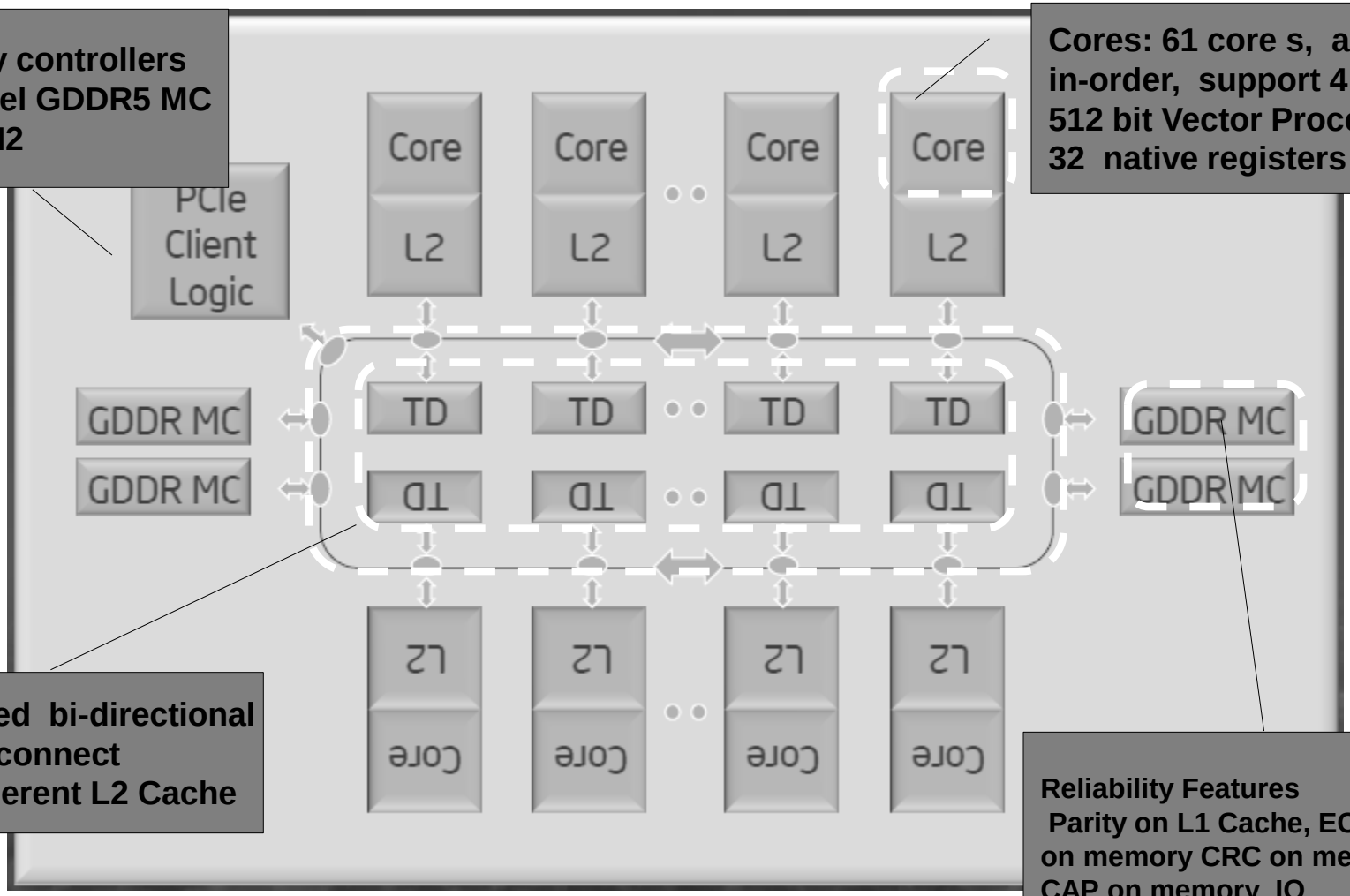
MIC Architecture, System Overview

Compute modes vision



Source : References & Intel Xeon-Phi; http://www.intel.com/

Intel® Xeon Phi™ Architecture Overview



8 memory controllers
16 Channel GDDR5 MC
PCIe GEN2

Cores: 61 core s, at 1.1 GHz
in-order, support 4 threads
512 bit Vector Processing Unit
32 native registers

High-speed bi-directional ring interconnect
Fully Coherent L2 Cache

Reliability Features
Parity on L1 Cache, ECC on memory CRC on memory IO, CAP on memory IO

Source : References & Intel Xeon-Phi; http://www.intel.com/

OpenMP : SIMD Constructs

`simd` construct

Summary

The **`simd`** construct can be applied to a loop to indicate that the loop can be transformed into a SIMD loop (that is, multiple iterations of the loop can be executed concurrently using SIMD instructions).

Syntax

C/C++

The syntax of the **`simd`** construct is as follows:

```
#pragma omp simd [clause[[,] clase] … ] new-line
        for-loops
```

Source : NVIDIA, PGI & References given in the presentation

OpenMP : SIMD Constructs

Syntax

C/C++

The syntax of the **`simd`** construct is as follows:

```
#pragma omp simd [clause[[,] clause] …] new-line
        for-loops
```

where *clause* is one of the following:

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

The **`simd`** directive places restrictions on the structure of the associated *for-loops*. Specifically, all associated *for-loops* must have canonical loop form

Source : http:/www.openmp.org; References of OpenMP

OpenMP : `declare simd` Construct

Summary

The **`declare simd`** construct can be applied to a function (C, C++ and Fortran) or a subroutine (Fortran) to enable the creation of one or more versions that can process multiple arguments using SIMD instructions from a single invocation from a SIMD loop. The **`declare simd`** directive is a declarative directive. There may be multiple **`declare simd`** directives for a function (C, C++, Fortran) or subroutine (Fortran 90).

Source : http:/www.openmp.org; References of OpenMP

OpenMP : `declare simd` Construct

Syntax

C/C++

The syntax of the **`declare simd`** construct is as follows:

```
#pragma omp declare simd [clause[[,] clause] …] new-line
/#pragma omp declare simd [clause[[,] clause] …] new-line
[…]
        function definition or declaration
```

where *clause* is one of the following:

- **`simdlen`***(length)*
- **`linear`***(argument-list[:constant-linear-step])*
- **`aligned`***(argument-list[:alignment])*
- **`private`***(argument-list)*
- **`inbranch`**
- **`notinbranch`**

Source : http:/www.openmp.org; References of OpenMP

OpenMP : `Loop SIMD` Constructs

Summary

The loop SIMD construct specifies a loop that can be executed concurrently using SIMD instructions and that those iterations will also be executed in parallel by threads in the team.

Syntax

C/C++

Description

The loop SIMD construct will first distribute the iterations of the associated loop(s) across the implicit tasks of the parallel region in a manner consistent with any clauses that apply to the loop construct. The resulting chunks of iterations will then be converted to a SIMD loop in a manner consistent with any clauses that apply to the **`simd`** construct. The effect of any clause that applies to both constructs is as if it were applied to both constructs separately.

Source : http:/www.openmp.org; References of OpenMP

OpenMP : `Device` Construct

Summary

Create a device data environment for the extent of the region.

Syntax

C/C++

The syntax of the **`target data`** construct is as follows:

```
#pragma omp target data [clause[[,] clase] … ] new-line
        structured-block
```

Where *clause* is one of the following:

`device`(integer-expression)

`map`(*[map-type :] list*))

`if` (*scalar-expression*)

OpenMP `Device` Construct

Binding

The binding task region for a **`target data`** construct is the encountering task. The target region binds to the enclosing parallel or task region.

Description

When a **`target data`** construct is encountered, a new device data environment is created, and the encountering task executes the **`target data`** region. If there is no **`device`** clause, the default device is determined by the *default-device-var* ICV. The new device data environment it constructed form the enclosing device data environment, the data environment of the encountering task and any data-mapping clauses on the construct. When an **`if`** clause is present and the **`if`** clause expression evaluates to false, the device is the host.

Restrictions

A program must not depend on any ordering of the valuations of the clauses of the **`target data`** directive, or any side effects of the evaluations of the clauses.

At most one **`device`** clause can appear on the directive. The **`device`** expression must evaluate to a non-negative integer value.

At most one **`if`** clause can appear on the directive

OpenMP `target` Construct

Summary

Create a device data environment and execute the construct on the same device.

Syntax

C/C++

The syntax of the **`target`** construct is as follows:

```
#pragma omp target data [clause[[,] clase] … ] new-line
        structured-block
```

Where *clause* is one of the following:

`device`(*integer-expression*)

`map`(*[map-type :] list*))

`if` (*scalar-expression*)

Source : http:/www.openmp.org; References of OpenMP

OpenMP `target` Construct

Binding

The binding task region for a **`target`** construct is the encountering task. The target region binds to the enclosing parallel or task region.

Description

When a **`target`** construct provides a superset of the functionality and restrictions provided by the **`target data`** directive. The functionality added to the **`target`** directive is the inclusion of an executable region to be executed by a device. That is, the **`target`** directive is an executable directive. The encountering task waits for the device to complete the target region. When an **`if`** clause is present and the **`if`** clause expression evaluated to *false*, the target region is executed by the host device.

Restrictions

- If a **`target, target update`**, or **`target data`** construct appears within a target region then the behaviour is unspecified.
- The result of an **`omp_set_default_device, omp_get_default_device`**.

OpenMP `target update` Construct

Summary

The **`target update`** directive makes the corresponding list item in the device data environment consistent with their original list items, according to the specified motion clauses. The **`target update`** construct is a stand-alone directive

Syntax

C/C++

The syntax of the **`target`** construct is as follows:

```
#pragma omp target data [clause[[,] clase] … ] new-line
```

Where *motion-clause* is one of the following:

`to`(*list*)

`from`(*list*)

and where clause is motion-clause or one of the of following:

`device`(*integer-expression*)

`from`(*scalar-expression*)

OpenMP `target update` Construct

Binding

The binding task for a **`target update`** construct is the encountering task. The **`target update`** directive is a stand-alone directive.

Description

For each list item in a **`to`** or **`from`** clause there is a corresponding list item and an original list item. If the corresponding list item is not present in the device data environment, the behaviour is unspecified. Otherwise, each corresponding list item in the device data environment has an original list item in the current task's data environment.

For each list item in a **`from`** clause the value of the corresponding list item is assigned to the original list item.

The list items that appear in the to or from clauses may include array sections.

The device is specified in the **`device`** clause. If there is no **`device`** clause, the device is determined by the *default-device-var* ICV. When an **`if`** clause is present and the **`if`** clause expression evaluates to false then no assignments occur.

Restrictions

- A program must bod depend on any ordering of the evaluations of the clauses of the **`target update`** directive, or on any side effects of the evaluations of the clauses.

OpenMP `Declare target` Directive

Summary

The **`declare target`** directive specifies that variables, functions (C, C++ and Fortran), and subroutines (Fortran) are mapped to a device. The **`declare target`** directive is a declarative directive.

Syntax

C/C++

The syntax of the **`declare target`** directive is as follows:

```
#pragma omp declare target new-line
declarations-definition-seq
#pragma omp end declare target new-line
```

Source : http:/www.openmp.org; References of OpenMP

Description

OpenMP `Declare target` Directive

C/C++

Variable and routine declarations that appear between the **`declare target`** and **`end declare target`** directives form an implicit list where each list item is the variable or function name.

C/C++

Fortran

If a declare target does not have an explicit list, then an implicit list of one item is formed from the name of the enclosing subroutine, subprogram, function subprogram or interface body to which it applies.

Fortran

If a list item is a function (C, C++, Fortran) subroutine (Fortran) then a device-specific version of the routine is created that can be called from a target region.

If a list item is a variable then the original variable is mapped to a corresponding variable in the initial deivce data environment for all devices. If the original variable is initialized the corresponding variable in the device data environment is initialized with the same value.

Restrictions

- A threadprivate variable cannot appear in a **`declare target`** directive
- A variable declared in a **`declare target`** directive must have a mappable type.

OpenMP teams Construct

Summary

The **teams** construct creates a league of thread teams and the master thread of each team executes the region.

Syntax

The syntax of the **team** construct is as follows:

C/C++

```
#pragma omp teams[clause[[,] clase] … ] new-line
structured-block
```

and where clause is one of the of following:

num_teams(*integer-expression*)

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

OpenMP teams Construct

Binding

The binding thread set for a **teams** region is the encountering thread.

Description

When a thread encounters a teams construct, a league of thread teams is created and the master thread of each thread team executes the teams region.

The number of teams created in implementation defined, but is less than or equal to the value specified in the **num_teams** clause.

The maximum number of threads participating in the contention group that each team initiates is implementation defined, but is less than or equal to the value specified in the **thread_limit** clause.

Once the **teams** are created, the number of teams remains constant for the duration of the teams region.

Within a teams region, team numbers uniquely identify each team. Team numbers are consecutive whole numbers ranging from zero to one less than the number of teams. A thread may obtain its own team number by a call to the **omp_get_team_num** library routine.

The threads other than the master thread do not begin execution until the master thread encounters a **parallel** region.

After the teams have completed execution of the **teams** region, the encountering thread resumes execution of the enclosing **target** region. There is no implicit barrier at the end of a **teams** construct.

OpenMP `distribute` Construct

Summary

The **`distribute`** construct specifies that the iterations of one or more loops will be executed by the thread teams in the context of their implicit tasks. The iterations are distributed across the master threads of all teams that execute the **`teams`** region to which the **`distribute`** region binds..

Source : http:/www.openmp.org; References of OpenMP

OpenMP `distribute` Construct

Syntax

C/C++

The syntax of the **`distribute`** construct is as follows:

```
#pragma omp distribute [clause[[,] clase] … ] new-line
      for-loops
```

Where *clause* is one of the following:

`private` (*list*)
`firstprivate` (*list*)
`collaspe` (*n*)
`dist_schedule` (*kind[, chunk_size]*)

All associated for – loops must have the canonical form described in Section 2.6 on page 51

OpenMP `distribute` Construct

Binding

The binding thread set for a **`distribute`** region is the set of master threads created by a **`teams`** construct. A **`distribute`** region binds to the innermost enclosing **`teams`** region. Only the threads executing the binding **`teams`** region participate in the execution of the loop iterations.

Description

The **`distribute`** construct is associated with a loop nest consisting of one or more loops that follow the directive.

There is no implicit barrier at the end of a **`distribute`** construct.

The **`collapse`** clause may be used to specify how many loops are associated with the **`distribute`** construct. The parameter of the **`collapse`** clause must be a constant positive integer expression. If no **`collapse`** clause is present, the only loop that is associated with the **`distribute`** construct is the one that immediately follows the **`distribute`** construct.

OpenMP `distribute` Construct

If more that one loop is associated with the **`distribute`** construct, then the iteration of all associated loops are collapsed into one larger iteration space. The sequential execution of the iterations in all associated loops determines the order of the iterations in the collapsed iteration space.

If **`dist_schedule`** is specified kind must be static. If specified, iterations are divided into chunks of size *chunk_size*, chunks are assigned to the teams of the league in a round-robin fashion in the order of the team number. When no *chunk_size* is specified, the iteration space is divided into chunks that are approximately equal in size, and at most one chunk is distributed to each team of the league. Note that the size of the chunks is unspecified in this case.

When no **`dist_schedule`** clause is specified, the schedule is implementation defined.

OpenMP `distribute simd` Construct

Summary

The **`distribute simd`** construct specifies a loop that will be distributed across the master threads of the teams region and executed concurrently using SIMD instructions.

Syntax

The syntax of the **`team`** construct is as follows:

C/C++

```
#pragma omp distribute simd[clause[[,] clase] … ]
      for-loops
```

Where clause can be any of the clauses accepted by the **`distribute`** or **`simd`** directives with identical meaning and restrictions:

C/C++

Fortran

```
!$omp distribute simd[clause[[,] clase] … ]
      do-loops
[ !$omp and distribute simd ]
```

OpenMP `distribute simd` Construct

Description

The **`distribute simd`** construct will first distribute the iterations of the associated loop(s) according to the semantics of the **`distribute`** construct and any clauses that apply to the distribute construct. The resulting chunks of iterations will then be converted to a SIMD loop in a manner consistent with any clauses that apply to the **`simd`** construct. The effect of any clause that applies to both constructs is as if it were applied to both constructs separately.

Restrictions

The restrictions for the **`distribute`** and **`simd`** constructs apply.

Cross References

- **`simd`** construct, see Section 2.8.1 on page 68
- **`distribute`** construct, see Section 2.9.6 on page 88
- Data attribute clauses

OpenMP Distribute Parallel Loop Construct

Summary

The distribute parallel loop construct specifies a loop that can be executed in parallel by multiple threads that are members of multiple teams.

Syntax

The syntax of the distribute parallel loop construct is as follows:

C/C++

```
#pragma omp distribute parallel for[clause[[,] clause] … ]
      for-loops
```

Where clause can be any of the clauses accepted by the **`distribute`** or parallel loop directives with identical meaning and restrictions:

C/C++

Source : http:/www.openmp.org; References of OpenMP

OpenMP `distribute simd` Construct

Description

The distribute parallel loop construct will first distribute the iterations of the associated loop(s) according to the semantics of the **`distribute`** construct and any clauses that apply to the **`distribute`** construct. The resulting loops will then be distributed across the threads contained within the **`teams`** region to which the **`distribute`** construct binds in a manner consistent with any clauses that apply to the parallel loop construct. The effect of any clause that applies to both the **`distribute`** and parallel loop constructs is as if it were applied to both constructs separately.

Restrictions

The restrictions for the **`distribute`** and parallel loop constructs apply.

Cross References

- **`distribute`** construct, Parellel loop construct, Data attribute clauses

OpenMP `Distribute Parallel Loop SIMD` Construct

Summary

The distribute parallel loop SIMD construct specifies a loop that can be executed in concurrently using SIMD instruction in parallel by multiple threads that are members of multiple teams.

Syntax

The syntax of the distribute parallel loop SIMD construct is as follows:

C/C++

```
#pragma omp distribute parallel for simd [clause[[,] clause] … ]
    for-loops
```

Where *clause* can be any of the clauses accepted by the **`distribute`** or parallel loop SIMD directives with identical meaning and restrictions:

C/C++

OpenMP `Distribute Parallel Loop SIMD` Construct

Description

The distribute parallel loop construct will first distribute the iterations of the associated loop(s) according to the semantics of the **`distribute`** construct and any clauses that apply to the **`distribute`** construct. The resulting loops will then be distributed across the threads contained within the **`teams`** region to which the **`distribute`** construct binds in a manner consistent with any clauses that apply to the parallel loop construct. The resulting chunks of iterations will then be converted to a SIMD loop in a manner consistent with any clauses that apply to the **`simd`** construct. The effect of any clause that applies to both the **`distribute`** and parallel loop SIMD constructs is as if it were applied to both constructs separately.

Source : http:/www.openmp.org; References of OpenMP

OpenMP Combined Construct

Description

Combined constructs are shortcuts for specifying one construct immediately nested inside another construct. The semantics of the combined constructs are identical to that of explicitly specifying the first construct containing one instance of the second construct and no other statements.

Some combined constructs have clauses that are permitted on both constructs that were combined. Where specified, the effect is as if applying the clauses to one or both constructs. If not specified and applying the clause to one to one construct would result in different program behaviour than applying the clause to the other construct then the program's behaviour is unspecified.

Source : NVIDIA, PGI & References given in the presentation

OpenMP `Parallel for` Loop Construct

Summary

The parallel loop construct is a shortcut for specifying a **`parallel`** construct containing one or more associated loops and no other statements.

Syntax

The syntax of the parallel loop construct is as follows:

C/C++

```
#pragma omp parallel for [clause[[,] clause] … ] new-line
        for-loops
```

where *clause* can be any of the clauses accepted by the **`parallel`** or **`for`** directives, except the **`nowait`** clause, with identical meanings and restrictions.

C/C++

Fortran

OpenMP `parallel section` Construct

Summary

The parallel sections construct is a shortcut for specifying a **`parallel`** construct containing one **`sections`** construct and no other statements.

Syntax

The syntax of the **`parallel sections`** construct is as follows:

C/C++

```
#pragma omp parallel for [clause[[,] clause] … ] new-line
      {
       [#pragma omp section new-line]
           structured-block
       [#pragma omp section new-line
           structured-block]
      …
      }
```

where *clause* can be any of the clauses accepted by the **`parallel`** or **`sections`** directives, except the **`nowait`** clause, with identical meanings and restrictions.

C/C++

OpenMP : `parallel workshare` Construct

Syntax

The syntax of the **`parallel workshare`** construct is as follows:

```
!$omp parallel for [clause[[,] clause]
        structured-block
!$omp end parallel workshare
```

where *clause* can be any of the clauses accepted by the **`parallel`** directives, with identical meanings and restrictions. **`nowait`** may not be specified on an **`end parallel workshare`** directive.

Description

The semantics are identical to explicitly specifying a **`parallel`** directive immediately followed by a **`workshare`** directive, and an **`end workshare`** directive immediately followed by an **`end parallel`** directive.

OpenMP `Parallel for SIMD` Loop Construct

Summary

The parallel loop SIMD construct is a shortcut for specifying a **`parallel`** construct containing one loop SIMD construct and no other statement.

Syntax

C/C++

```
#pragma omp parallel for simd [clause[[,] clause] … ] new-line
        for-loops
```

where *clause* can be any of the clauses accepted by the **`parallel`**, **`for`** or **`simd`** directives, except the **`nowait`** clause, with identical meanings and restrictions.

C/C++

Fortran

OpenMP `Parallel for SIMD` Loop Construct

Summary

The semantics of the parallel loop SIMD construct are identical to explicitly specifying a **`parallel`** directive immediately followed by a loop SIMD directive. The effect of any clause that applies to both constructs is as if it were applied to the loop SIMD construct and not to the **`parallel`** construct.

Source : NVIDIA, PGI & References given in the presentation

OpenMP `target teams` Construct

Summary

The target teams construct is a shortcut for specifying a **`target`** construct containing a **`teams`** construct

Syntax

The syntax of the target teams construct is as follows:

C/C++

```
#pragma omp parallel for [clause[[,] clause] … ]
      structured-block
```

where *clause* can be any of the clauses accepted by the **`target`** or **`teams`** directives with identical meanings and restrictions

C/C++

OpenMP `teams distribute` Construct

Summary

The **`teams distribute`** construct is a shortcut for specifying a **`team`** construct containing a **`distribute`** construct

Syntax

The syntax of the **`teams distribute`** construct is as follows:

C/C++

```
#pragma omp team distribute [clause[[,] clause] … ]
        for-loops
```

where *clause* can be any of the clauses accepted by the **`teams`** or **`distribute`** directives with identical meanings and restrictions

C/C++

OpenMP `teams distribute simd` Construct

Summary

The **`teams distribute simd`** construct is a shortcut for specifying a **`teams`** construct containing a **`distribute simd`** construct

Syntax

The syntax of the **`teams distribute simd`** construct is as follows:

C/C++

```
#pragma omp team distribute [clause[[,] clause] … ]
        for-loops
```

where *clause* can be any of the clauses accepted by the **`teams`** or **`distribute simd`** directives with identical meanings and restrictions

C/C++

Fortran

```
!$omp teams distribute simd [clause[[,] clause] … ]
        for-loops
[!$ompand end teams distribute simd]
```

OpenMP `teams distribute simd` Construct

`!$omp teams distribute simd` *[clause[[,] clause] …]*
do-loops
*[*__`!$ompand end teams distribute simd`__*]*

where *clause* can be any of the clauses accepted by the **`teams`** or **`distribute simd`** directive with identical meanings and restrictions.

If an **`end teams distribute`** directive is not specified, an **`end teams distribute`** directive is assumed at the end of the *do-loops*.

Fortran

Description

The semantics are identical to explicitly specifying a **`teams`** directive immediately followed by a **`distribute simd`** directive. Some clauses are permitted on both constructs.

Source : http:/www.openmp.org; References of OpenMP

OpenMP `target teams distribute` Construct

Summary

The **`target teams distribute`** construct is a shortcut for specifying a **`target`** construct containing a **`teams distribute`** construct

Syntax

The syntax of the **`target teams distribute`** construct is as follows:

C/C++

```
#pragma omp target team distribute [clause[[,] clause] … ]
      for-loops
```

where *clause* can be any of the clauses accepted by the **`target`** or **`team distribute`** directives with identical meanings and restrictions

C/C++

OpenMP `target teams distribute simd` Construct

Summary

The **`target teams distribute`** construct is a shortcut for specifying a **`target`** construct containing a **`teams distribute`** construct

Syntax

The syntax of the **`target teams distribute`** construct is as follows:

C/C++

```
#pragma omp target teams distribute simd [clause[[,] clause] … ]
      for-loops
```

where *clause* can be any of the clauses accepted by the **`target`** or **`team distribute simd`** directives with identical meanings and restrictions

C/C++

OpenMP `teams distribute parallel for` Construct

Summary

The teams distribute parallel loop construct is a shortcut for specifying a **`teams`** construct containing a distribute parallel loop construct.

Syntax

The syntax of the teams distribute parallel loop construct is as follows:

C/C++

```
#pragma omp teams distribute parallel for [clause[[,] clause] … ]
      for-loops
```

where *clause* can be any of the clauses accepted by the **`teams`** or **`distribute parallel for`** directives with identical meanings and restrictions

C/C++

Source : http:/www.openmp.org; References of OpenMP

OpenMP : Target Teams Distribute Parallel Loop Construct

Summary

The target teams distribute parallel loop construct is a shortcut for specifying a **`target`** construct containing a teams distribute parallel loop construct.

Syntax

The syntax of the target teams distribute parallel loop construct is as follows:

C/C++

```
#pragma omp target teams distribute parallel for [clause[[,] clause] … ]
    for-loops
```

where *clause* can be any of the clauses accepted by the **`target`** or **`teams distribute parallel for`** directives with identical meanings and restrictions

C/C++

Source : http:/www.openmp.org; References of OpenMP

OpenMP : Target Teams Distribute Parallel Loop Construct

Summary

The **`teams distribute`** parallel construct is a shortcut for specifying a **`teams`** construct containing a distribute parallel loop SIMDconstruct

Syntax

The syntax of the **`teams distribute simd`** construct is as follows:

C/C++

```
#pragma omp team distribute [clause[[,] clause] … ]
      for-loops
```

where *clause* can be any of the clauses accepted by the **`teams`** or **`distribute parallel for simd`** directives with identical meanings and restrictions

C/C++

Fortran

```
!$omp teams distribute parallel do simd [clause[[,] clause] … ]
      for-loops
[!$omp and end teams distribute parallel do simd]
```

OpenMP : Target Teams Distribute Parallel Loop Construct

Summary

The **`teams distribute`** parallel construct is a shortcut for specifying a **`teams`** construct containing a distribute parallel loop SIMD construct

Syntax

The syntax of the **`teams distribute simd`** construct is as follows:

C/C++

```
#pragma omp team distribute parallel for simd [clause[[,] clause] … ]
        for-loops
```

where *clause* can be any of the clauses accepted by the **`teams`** or **`distribute parallel for simd`** directives with identical meanings and restrictions

C/C++

Source : http:/www.openmp.org; References of OpenMP

OpenMP:Target Teams Distribute Parallel Loop SIMD Construct

Summary

The target **`teams`** distribute parallel loop SIMD construct is a shortcut for specifying a **`target`** construct containing a teams distribute parallel loop SIMD construct.

Syntax

The syntax of the target teams distribute parallel loop SIMD construct is as follows:

C/C++

```
#pragma omp team distribute [clause[[,] clause] … ]
        for-loops
```

where *clause* can be any of the clauses accepted by the **`target`** or **`teams distribute parallel for simd`** directives with identical meanings and restrictions

C/C++

OpenMP Tasking Construct

Summary

The **`teams`** construct defines an explicit task.

Syntax

The syntax of the target teams distribute parallel loop SIMD construct is as follows:

C/C++

```
#pragma omp task [clause[[,] clause] … ] new-line
        structured-block
```

where *clause* is one of the following:

`if` (*scalar-expression*)
`final` (*scalar-expression*)
`untied`
`default`(**`shared | none`**)
`mergeable`
`private` (*list*)
`firstprivate` (*list*)
`shared` (*list*)
`depend` (*dependence-type : list*)

OpenMP :Tasking Construct

Description

The encountering thread may immediately execute the task, or defer its execution. In the latter case, any thread in the team may be assigned the task. Completion of the task can be guaranteed using task synchronization constructs. A **`task`** construct may be nested inside an outer task, but the **`task`** region of the inner task is not a part of the **`task`** region of the outer task.

When an **`if`** clause is present on a **`task`** construct, and the **`if`** clause expression evaluates to *false*, an undeferred task is generated, and the encountering thread must suspend the current task region, for which execution cannot be resumed until the generated task is completed. Note that the use of a variable in an **`if`** clause expression of a **`task`** construct causes an implicit reference to the variable in all enclosing constructs.

When a **`final`** clause is present on a **`task`** construct and the **`final`** clause expression evaluates to true, the generated task will be a final task. All **`task`** constructs encountered during execution of a final task will generate final and included tasks. Note that the use of a variable in a **`final`** clause expression of a **`task`** construct cause an implicit reference to the variable in all enclosing constructs.

OpenMP : `depend` Clause

Summary

The **`depend`** clause enforces additional constraints on the scheduling of tasks. These constraints establish dependences only between sibling tasks. The clause consists of a *dependence-type* with one or more list items.

Syntax

The syntax of the target teams distribute parallel loop SIMD construct is as follows:

```
depend( dependence-type : list )
```

Description

Task dependences are derived from the dependence-type of a **`depend`** clause and its list items, where *dependence-type* is one of the following:

OpenMP : `depend` Clause

Description

Task dependences are derived from the dependence-type of a **`depend`** clause and its list items, where *dependence-type* is one of the following:

The **`in`** *dependence-type*. The generated task will be a dependent task of all previously generated sibling tasks that reference at least one of the list items in an **`out`** or **`inout`** dependence-type list.

The **`out`** and **`inout`** dependence-types. The generated task will be a dependent task of all previously generated sibling tasks that reference at least one of the list items in an **`in`**, **`out`**, or **`inout`** dependence-type list.

The list items that appear in the **`depend`** clause may include array sections.

Note – The enforced task dependence establishes a synchronization of memory accesses performed by a dependent task with respect to accesses performed by the predecessor tasks. However, it is the responsibility of the programmer to synchronize properly with respect to other concurrent accesses that occur outside of those tasks.

OpenMP : `taskyield` Clause

Summary

The **`taskyield`** construct specifies that the current task can be suspended in favour of execution of a different task. The taskyield construct is a stand-along directive.

Syntax

C/C++

The syntax of the **`taskyield`** construct is as follows:

```
#pragma omp taskyield new-line
```

C/C++

Fortran

The syntax of the **`taskyield`** construct is as follows:

```
!$omp taskyield
```

OpenMP : `task Scheduling` Clause

Whenever a thread reaches a task scheduling point, the implementation may cause it to perform a task switch, beginning or resuming execution of a different task bound to the current team. Task scheduling points are implied as the following locations:

- the point immediately following the generation of an explicit **`task`** region
- in a **`taskyield`** region
- in a **`taskwait`** region
- at the end of a **`taskgroup`** region
- in an implicit and explicit **`barrier`** region
- the point immediately following the generation of a **`target`** region
- at the beginning and end of a **`target data`** region
- in a **`target update`** region

Source : http:/www.openmp.org; References of OpenMP

OpenMP : Data Environment

This section presents a directive and several clauses for controlling the data environment during the execution of **`parallel`**, **`task`**, **`simd`**, and worksharing regions.

- how the data-sharing attributes of variables referenced in **`parallel, task, simd`**, and worksharing regions are determined.
- The **`threadprivate`** directive, which is provided to create threadprivate memory
- Clauses that may be specified on directives to control the data-sharing attributes of variables referenced in **`parallel, task, simd`** or worksharing constructs
- Clauses that may be specified on directives to copy data values from private or threadprivate variables on one thread to the corresponding variables on other threads in the team
- Clauses that may be specified on directives to map variables to devices

OpenMP : `threadprivate` Directive

Summary

The **`threadprivate`** directive specifies that variable are replicated, with each thread having its own copy. The **`threadprivate`** directive is a declarative directive.

Syntax

C/C++

The syntax of the **`threadprivate`** directive is as follows:

```
#pragma omp threadprivate (list) new-line
```

Where list is a comma-separated list of file-scope, namespace-scope, or static block-scope variables that do not have incomplete types.

C/C++

OpenMP : Data-Sharing Attribute Clauses

Several constructs accept clauses that allow a user to control the data-sharing attributes of variables referenced in the construct. Data-sharing attribute clauses apply only to variables for which the names are visible in the construct on which the clause appears.

Not all of the clauses listed in this section are valid on all directives. The set of clauses that is valid on a particular directive is described with the directive.

Most of the clauses accept a comma-separated list of list items (see Section 2.1 on age 26). All list items appearing in a clause must be visible, according toe the scoping rules of the base language. With the exception of the default clause, clauses may be repeated as needed. A list item that specifies a give variable may not appear in more than one clause on the same directive, except that a variable may be specified in both firstprivate and lastprivate clauses.

C/C++

If a variable referenced in a data-sharing attribute clause has a type derived from a template, and there are no other references to that variable in the program, then any behaviour related to that variable is unspecified

C/C++

OpenMP : Data Copying Clauses

This section describes the **`copyin`** clause (allowed on the **`parallel`** directive and combined parallel worksharing directives) and the **`copyprivate`** clause (allowed on the **`single`** directive).

These clauses support the copying of data values from private or threadprivate variables on one implicit task or thread to the corresponding variables on other implicit tasks or threads in the team.

The clauses accept a comma-seperated list of list items (see Section 2.1 on page 26). All list items appearing in a clause must be visible, according to the scoping rules of the base language. Clauses may be repeated as needed, but a list item that specifies a given variable may not appear in more than one clause on the same directive.

Source : http:/www.openmp.org; References of OpenMP

OpenMP : `copypriavate` clause

Summary

The **`copyprivate`** clause provides a mechanism to use a private variable to broadcast a value from the data environment of one implicit task to be data environments of the other implicit tasks belonging to the **`parallel`** region.

To avoid race conditions, concurrent reads or updates of the list item must be synchronized with the update of the list item that occurs as a result of the **`copyprivate`** clause.

Syntax

The syntax of the **`copyprivate`** clause is as follows:

```
copyprivate (list)
```

Description

The effect of the **`copypriate`** clause on the specified list items occurs after the execution of the structured block associated with the single construct (see Section 2.7.3 on page 63), and before any of the threads in the team have left the barrier at the end of the construct.

OpenMP : `map` clause

Summary

The **`map`** clause maps a variable from the current task's data environment to the device data environment associated with the construct.

Syntax

The syntax of the **`copyprivate`** clause is as follows:

```
map (list)
```

Description

The list items that appear in a **`map`** clause may include array sections.

For list items that appear in a **`map`** clause, corresponding new list items are created in the device data environment associated with the construct.

The original and corresponding list items may share storage such that write to either item by one task followed by a read or write of the other item by another task without intervening synchronization can result in data races.

OpenMP : `declare reduction` Directive

Summary

The following section describes the directive for declaring user-defined reductions. The **`declare reduction`** directive declares a reduction-identifier that can be used in **`reduction`** clause. The **`declare reduction`** directive is a declarative directive.

Syntax

C/C++

`#pragma omp declare reduction` *(reduction-identifier : typename-list : combiner) [initializer-clause] new-line*

where:

- *reduction-identifier* is either a base language identifier or one of the following operators **`+, -, *, &, |, ^, &&`** and **`||`**
- *typename-list* in list of type names
- *combiner* is an expression
- *Initializer-clause* is initializer (*initializer-expr*) where i*nitializer-expr* is omp_priv * *initializer* or *function-name* (*argument-list*)

C

Conclusions

- Discussed An overview of Programming for Multi-Core Systems with Coprocessors OpenMP 4.0

Source : OpenMP Web Sites and References

References & Acknowledgements

References :

1. Theron Voran, Jose Garcia, Henry Tufo, University Of Colorado at Boulder National Center or Atmospheric Research, TACC-Intel Hihgly Parallel Computing Symposium, Austin TX, April 2012
2. Robert Harkness, Experiences with ENZO on the Intel R Many Integrated Core (Intel MIC) Architecture, National Institute for Computational Sciences, Oak Ridge National Laboratory
3. Ryan C Hulguin, National Institute for Computational Sciences, Early Experiences Developing CFD Solvers for the Intel Many Integrated Core (Intel MIC) Architecture, TACC-Intel Highly Parallel Computing Symposium April, 2012
4. Scott McMillan, Intel Programming Models for Intel Xeon Processors and Intel Many Integrated Core (Intel MIC) Architecture, TACC-Highly Parallel Comp. Symposium April 2012
5. Sreeram Potluri, Karen Tomko, Devendar Bureddy , Dhabaleswar K. Panda, Intra-MIC MPI Communication using MVAPICH2: Early Experience, Network-Based Computing Laboratory, Department of Computer Science and Engineering The Ohio State University, Ohio Supercomputer Center, TACC-Highly Parallel Computing Symposium April 2012
6. Karl W. Schulz, Rhys Ulerich, Nicholas Malaya ,Paul T. Bauman, Roy Stogner, Chris Simmons, Early Experiences Porting Scientific Applications to the Many Integrated Core (MIC) Platform ,Texas Advanced Computing Center (TACC) and Predictive Engineering and Computational Sciences (PECOS) Institute for Computational Engineering and Sciences (ICES), The University of Texas at Austin ,Highly Parallel Computing Symposium ,Austin, Texas, April 2012
7. Kevin Stock, Louis-Noel, Pouchet, P. Sadayappan ,Automatic Transformations for Effective Parallel Execution on Intel Many Integrated, The Ohio State University, April 2012
8. http://www.tacc.utexas.edu/
9. Intel MIC Workshop at C-DAC, Pune April 2013
10. First Intel Xeon Phi Coprocessor Technology Conference iXPTC 2013 New York, March 2013
11. Shuo Li, Vectorization, Financial Services Engineering, software and Services Group, Intel ctel Corporation;
12. Intel® Xeon Phi™ (MIC) Parallelization & Vectorization, Intel Many Integrated Core Architecture, Software & Services Group, Developers Relations Division

References & Acknowledgements

References :

13. Intel® Xeon Phi™ (MIC) Programming, Rama Malladi, Senior Application Engineer, Intel Corporation, Bengaluru India April 2013
14. Intel® Xeon Phi™ (MIC) Performance Tuning, Rama Malladi, Senior Application Engineer, Intel Corporation, Bengaluru India April 2013
15. Intel® Xeon Phi™ Coprocessor Architecture Overview, Dhiraj Kalamkar, Parallel Computing Lab, Intel Labs, Bangalore
16. Changkyu Kim,Nadathur Satish ,Jatin Chhugani ,Hideki Saito,Rakesh Krishnaiyer ,Mikhail Smelyanskiy ,Milind Girkar, Pradeep Dubey, Closing the Ninja Performance Gap through Traditional Programming and Compiler Technology , Technical Report Intel Labs , Parallel Computing Laboratory , Intel Compiler Lab, 2010
17. Colfax International Announces Developer Training for Intel® Xeon Phi™ Coprocessor, Industry First Training Program Developed in Consultation with Intel SUNNYVALE, CA, Nov, 2012
18. Andrey Vladimirov Stanford University and Vadim Karpusenko , Test-driving Intel® Xeon Phi™ coprocessors with a basic N-body simulation Colfax International January 7, 2013 Colfax International, 2013 http://research.colfaxinternational.com/
19. Jim Jeffers and James Reinders,Intel® Xeon Phi™ Coprocessor High-Performance Programming by Morgann Kauffman Publishers Inc, Elsevier, USA. 2013
20. Michael McCool, Arch Robison, James Reinders, Structured Parallel Programming: Patterns for Efficient Computation, Morgan Kaufman Publishers Inc, 2013.
21. Dan Stanzione, Lars Koesterke, Bill Barth, Kent Milfeld by Preparing for Stampede: Programming Heterogeneous Many-Core Supercomputers. TACC, XSEDE 12 July 2012
22. John Michalakes, Computational Sciences Center, NREL, & Andrew Porter, Opportunities for WRF Model Acceleration, WRF Users workshop, June 2012
23. Jim Rosinski , Experiences Porting NOAA Weather Model FIM to Intel MIC, ECMWF workshop On High Performance Computing in Meteorology, October 2012
24. Michaela Barth, KTH Sweden , Mikko Byckling, CSC Finland, Nevena Ilieva, NCSA Bulgaria, Sami Saarinen, CSC Finland, Michael Schliephake, KTH Sweden, Best Practice Guide Intel Xeon Phi v0.1, Volker Weinberg (Editor), LRZ Germany March 31 ,2013

References & Acknowledgements

References :

25. Barbara Chapman, Gabriele Jost and Ruud van der Pas, Using OpenMP, MIT Press Cambridge, 2008
26. Peter S Pacheco, An Introduction Parallel Programming, Morgann Kauffman Publishers Inc, Elsevier, USA. 2011
27. Intel Developer Zone: Intel Xeon Phi Coprocessor,
28. http://software.intel.com/en-us/mic-developer
29. Intel Many Integrated Core Architecture User Forum,
30. http://software.intel.com/en-us/forums/intel-many-integrated-core
31. Intel Developer Zone: Intel Math Kernel Library, http://software.intel.com/en-us
32. Intel Xeon Processors & Intel Xeon Phi Coprocessors – Introduction to High Performance Applications Development for Multicore and Manycore – Live Webinar, 26.-27, February .2013,
33. recorded http://software.intel.com/en-us/articles/intel-xeon-phi-training-m-core
34. Intel Cilk Plus Home Page, http://cilkplus.org/
35. James Reinders, Intel Threading Building Blocks (Intel TBB), O'REILLY, 2007
36. Intel Xeon Phi Coprocessor Developer's Quick Start Guide,
37. http://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-developers-quick-start-guide
38. Using the Intel MPI Library on Intel Xeon Phi Coprocessor Systems,
39. http://software.intel.com/en-us/articles/using-the-intel-mpi-library-on-intel-xeon-phi-coprocessor-systems
40. An Overview of Programming for Intel Xeon processors and Intel Xeon Phi coprocessors,
41. http://software.intel.com/sites/default/files/article/330164/an-overview-of-programming-for-intel-xeon-processors-and-intel-xeon-phi-coprocessors_1.pdf
42. Programming and Compiling for Intel Many Integrated Core Architecture,
43. http://software.intel.com/en-us/articles/programming-and-compiling-for-intel-many-integrated-core-architecture
44. Building a Native Application for Intel Xeon Phi Coprocessors,
45. http://software.intel.com/en-us/articles/

References & Acknowledgements

References :

46. Advanced Optimizations for Intel MIC Architecture, http://software.intel.com/en-us/articles/advanced-optimizations-for-intel-mic-architecture
47. Optimization and Performance Tuning for Intel Xeon Phi Coprocessors - Part 1: Optimization Essentials, http://software.intel.com/en-us/articles/optimization-and-performance-tuning-for-intel-xeonphi-coprocessors-part-1-optimization
48. Optimization and Performance Tuning for Intel Xeon Phi Coprocessors, Part 2: Understanding and Using Hardware Events, http://software.intel.com/en-us/articles/optimization-and-performance-tuning-for-intel-xeon-phi-coprocessors-part-2-understanding
49. Requirements for Vectorizable Loops,
50. http://software.intel.com/en-us/articles/requirements-for-vectorizable-
51. R. Glenn Brook, Bilel Hadri, Vincent C. Betro, Ryan C. Hulguin, Ryan Braby. Early Application Experiences with the Intel MIC Architecture in a Cray CX1, National Institute for Computational Sciences. University of Tennessee. Oak Ridge National Laboratory. Oak Ridge, TN USA
52. http://software.intel.com/mic-developer
53. Loc Q Nguyen , Intel Corporation's Software and Services Group , Using the Intel® MPI Library on Intel® Xeon Phi™ Coprocessor System,
54. Frances Roth, System Administration for the Intel® Xeon Phi™ Coprocessor, Intel white Paper
55. Intel® Xeon Phi™ Coprocessor, James Reinders, Supercomputing 2012 Presentation
56. Intel® Xeon Phi™ Coprocessor Offload Compilation, Intel software

References & Acknowledgements

References

57. Andrews, Grogory R. (2000), Foundations of Multithreaded, Parallel, and Distributed Programming, Boston, MA : Addison-Wesley
58. Butenhof, David R (1997), Programming with POSIX Threads , Boston, MA : Addison Wesley Professional
59. Culler, David E., Jaswinder Pal Singh (1999), Parallel Computer Architecture - A Hardware/Software Approach , San Francsico, CA : Morgan Kaufmann
60. Grama Ananth, Anshul Gupts, George Karypis and Vipin Kumar (2003), Introduction to Parallel computing, Boston, MA : Addison-Wesley
61. Intel Corporation, (2003), Intel Hyper-Threading Technology, Technical User's Guide, Santa Clara CA : Intel Corporation Available at : http://www.intel.com
62. Shameem Akhter, Jason Roberts (April 2006), Multi-Core Programming - Increasing Performance through Software Multi-threading , Intel PRESS, Intel Corporation,
63. Bradford Nichols, Dick Buttlar and Jacqueline Proulx Farrell (1996), Pthread Programming O'Reilly and Associates, Newton, MA 02164,
64. James Reinders, Intel Threading Building Blocks – (2007) , O'REILLY series
65. Laurence T Yang & Minyi Guo (Editors), (2006) High Performance Computing - Paradigm and Infrastructure Wiley Series on Parallel and Distributed computing, Albert Y. Zomaya, Series Editor
66. Intel Threading Methodology ; Principles and Practices Version 2.0 copy right (March 2003), Intel Corporation
67. William Gropp, Ewing Lusk, Rajeev Thakur **(1999),** Using MPI-2, Advanced Features of the Message-Passing Interface, The MIT Press..
68. Pacheco S. Peter, **(1992),** Parallel Programming with MPI, , University of Sanfrancisco, Morgan Kaufman Publishers, Inc., Sanfrancisco, California
69. Kai Hwang, Zhiwei Xu, (**1998**), Scalable Parallel Computing (Technology Architecture Programming), McGraw Hill New York.
70. Michael J. Quinn (**2004**), Parallel Programming in C with MPI and OpenMP McGraw-Hill International Editions, Computer Science Series, McGraw-Hill, Inc. Newyork
71. Andrews, Grogory R. **(2000),** Foundations of Multithreaded, Parallel, and Distributed Progrmaming, Boston, MA : Addison-Wesley

References & Acknowledgements

References

72. SunSoft. Solaris multithreaded programming guide. SunSoft Press, Mountainview, CA, **(1996),** Zomaya, editor. Parallel and Distributed Computing Handbook. McGraw-Hill,
73. Chandra, Rohit, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, and Ramesh Menon, **(2001)**,Parallel Programming in OpenMP San Fracncisco Moraan Kaufmann
74. S.Kieriman, D.Shah, and B.Smaalders **(1995),** Programming with Threads, SunSoft Press, Mountainview, CA. 1995
75. Mattson Tim, **(2002),** Nuts and Bolts of multi-threaded Programming Santa Clara, CA : Intel Corporation, Available at : http://www.intel.com
76. I. Foster **(1995,** Designing and Building Parallel Programs ; Concepts and tools for Parallel Software Engineering, Addison-Wesley (1995
77. J.Dongarra, I.S. Duff, D. Sorensen, and H.V.Vorst **(1999),** Numerical Linear Algebra for High Performance Computers (Software, Environments, Tools) SIAM, 1999
78. OpenMP C and C++ Application Program Interface, Version 1.0". **(1998),** OpenMP Architecture Review Board. October 1998
79. D. A. Lewine. *Posix Programmer's Guide:* **(1991),** Writing Portable Unix Programs with the Posix. 1 Standard. O'Reilly & Associates, 1991
80. Emery D. Berger, Kathryn S McKinley, Robert D Blumofe, Paul R.Wilson, *Hoard : A Scalable Memory Allocator for Multi-threaded Applications* ; The Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX). Cambridge, MA, November (**2000**). Web site URL : http://www.hoard.org/
81. Marc Snir, Steve Otto, Steyen Huss-Lederman, David Walker and Jack Dongarra, (**1998**) *MPI-The Complete Reference: Volume 1, The MPI Core, second edition* [MCMPI-07].
82. William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir (**1998**) *MPI-The Complete Reference: Volume 2, The MPI-2 Extensions*
83. A. Zomaya, editor. Parallel and Distributed Computing Handbook. McGraw-Hill, **(1996)**
84. OpenMP C and C++ Application Program Interface, Version 2.5 (**May 2005**)", From the OpenMP web site, URL **: http://www.openmp.org/**
85. Stokes, Jon 2002 Introduction to Multithreading, Super-threading and Hyper threading *Ars Technica*, October **(2002)**

References & Acknowledgements

References

86. Andrews Gregory R. 2000, Foundations of Multi-threaded, Parallel and Distributed Programming, Boston MA : Addison – Wesley (**2000)**
87. Deborah T. Marr , Frank Binns, David L. Hill, Glenn Hinton, David A Koufaty, J . Alan Miller, Michael Upton, "Hyperthreading, Technology Architecture and Microarchitecture", Intel (**2000-01)**
88. http://www.erc.msstate.edu/mpi
89. http://www.arc.unm.edu/workshop/mpi/mpi.html
90. http://www.mcs.anl.gov/mpi/mpich
91. The MPI home page, with links to specifications for MPI-1 and MPI-2 standards : http://www.mpi–forum.org
92. Hybrid Programming Working Group Proposals, Argonne National Laboratory, Chiacago (2007-2008)
93. TRAC Link : https://svn.mpi-forum.org/trac/mpi-form-web/wiki/MPI3Hybrid
94. Threads and MPI Software, Intel Software Products and Services 2008 - 2009
95. Sun MPI 3.0 Guide November 2007
96. Treating threads as MPI processes thru Registration/deregistration –Intel Software Products and Services 2008 – 2009
97. Intel MPI library 3.2 - http://www.hearne.com.au/products/Intelcluster/edition/mpi/663/
98. http://www.cdac.in/opecg2009/
99. PGI Compilers http://www.pgi.com

References & Acknowledgements

References

100. Andrews Gregory R. 2000, Foundations of Multi-threaded, Parallel and Distributed Programming, Boston MA : Addison – Wesley (**2000)**
101. Deborah T. Marr , Frank Binns, David L. Hill, Glenn Hinton, David A Koufaty, J . Alan Miller, Michael Upton, "Hyperthreading, Technology Architecture and Microarchitecture", Intel (**2000-01)**
102. http://www.nvidia.com/object/nvidia-kepler.html NVIDIA Kepler Architecture 2012
103. http://developer.nvidia.com/cuda-toolkit NVIDIA CUDA toolkit 5.0 Preview Release April 2012
104. http://developer.nvidia.com/category/zone/cuda-zone NVIDIA Developer Zone
105. http://developer.nvidia.com/gpudirect RDMA for NVIDIA GPUDirect coming in CUDA 5.0 Preview Release, April 2012
106. http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf NVIDIA CUDA C Programming Guide Version 4.2 dated 4/16/2012 (April 2012)
107. http://developer.download.nvidia.com/assets/cuda/files/CUDADownloads/TechBrief_Dynamic_Parallelism_in_CUDA.pdf Dynamic Parallelism in CUDA Tesla K20 Kepler GPUs - Prelease of NVIDIA CUDA 5.0
108. http://developer.nvidia.com/cuda-downloads NVIDIA Developer ZONE - CUDA Downloads CUDA TOOLKIT 4.2
109. http://developer.nvidia.com/gpudirect NVIDIA Developer ZONE – GPUDirect
110. http://developer.nvidia.com/openacct OpenACC – NVIDIA
111. http://developer.nvidia.com/cuda-toolkit Nsight, Eclipse Edition Pre-release of CUDA 5.0, April 2012
112. The OpenCL Specification, Version 1.1,</B> Published by Khronos OpenCL Working Group, Aaftab Munshi (ed.), 2010.
113. NVIDA CUDA C Programming Guide Version V4.0, May 2012 (5/6/2012) http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf
114. http://www.khronos.org/files/opencl-1-1-quick-reference-card.pdf The OpenCL 1.1 Quick Reference card.
115. NVIDIA Developer Zone http://developer.nvidia.com/category/zone/cuda-zone
116. NVIDIA CUDA Toolkit 4.0 (May 2012) http://developer.nvidia.com/cuda-toolkit-4.0

References & Acknowledgements

References

117. Randi J. Rost, OpenGL – shading Language, Second Edition, Addison Wesley 2006
118. GPGPU Reference http://www.gpgpu.org
119. NVIDIA http://www.nvidia.com
120. NVIDIA tesla; http://www.nvidia.com/object/tesla_computing_solutions.html
121. NVIDIA CUDA Reference; http://www.nvidia.com/object/cuda_home.html
122. CUDA sample source code: http://www.nvidia.com/object/cuda_get_samples.html
123. href://www.nvidia.com/object/cuda_learn_products.html List of NVIDIA GPUs compatible with CUDA:
124. Download the CUDA SDK: www.nvidia.com/object/cuda_get.html
125. Specifications of nVIDIA GeForce 8800 GPUs:
126. RAPIDMIND http://www.rapidmind.net
127. Peak Stream - Parallel Processing (Acquired by Google in 2007) http:/www.google.co
128. http://www.guru3d.com/news/sandra-2009-gets-gpgpu-support/
129. AMD http:www.amd.com
130. AMD Stream Processors http://ati.amd.com/products/streamprocessor/specs.html
131. RAPIDMIND & AMD http://www.rapidmind.net/News-Aug4-08-SIGGRAPH.php
132. http://www-graphics.stanford.edu/projects/brookgpu/ Merrimac - Stream Arch. Standford Brook for GPUs
133. Standford : Merrimac - Stream Architecture http://merrimac.stanford.edu/
134. ATI RADEON - AMD http://www.canadacomputers.com/amd/radeon/
135. ATI & AMD - Technology Products http://ati.amd.com/products/index.html
136. Sparse Matrix Solvers on the GPU ; conjugate Gradients and Multigrid by Jeff Bolts, Ian Farmer, Eitan Grinspum, Peter Schroder , Caltech Report (2003); Supported in part by NSF, nVIDIA, etc....
137. Scan Primitives for GPU Computing by Shubhabrata Sengupta, Mark Harris*, Yao Zhang and John D Owens University of California Davis & *nVIDIA Corporation Graphic Hardware (2007).
138. Horm D; Stream reduction operations for GPGPU applciations in GPU Genes 2 Phar M., (Ed.) Addison Weseley, March 2005; Chapter 36, pp. 573-589 Graphic Hardware (2007).
139. Bollz J., Farmer I., Grinspun F., Schroder F : Sparse Matris Solvers on the GPU ; Conjugate Gradients and multigrid ACM Transactions on Graphics (Proceedings of ACM SIGRAPH 2003) 22, 2 (Jul y2003) pp 917-924 Graphic Hardware (2007).
140. NVIDIA CUDA Compute Unified Device Architecture – Prog. Guide - Ver1.1 November 2007

References & Acknowledgements

References

141. Tom R. Halfhill, *Number crunching with GPUs PeakStream Math API Exploits Parallelism in Graphics Processors, Ocotober 2006; Microprocessor* http://www.mdronline.com
142. Tom R. Halfhill, *Parallel Processing with CUDA Nvidia's High-Performance Computing Platform Uses Massive Multithreading ;* Microprocessors, Volume 22, Archive 1, January 2008 http://www.mdronline.com
143. J. Tolke, M.Krafczyk *Towards Three-dimensional teraflop CFD Computing on a desktop PC using graphics hardware Institute for Computational Modeling in Civil Engineering,* TU Braunschweig (2008)
144. I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Hoston, P.Hanrahan, Brook for GPUs ; *Stream Computing on GRaphics Hadrware, ACM Tran. GRaph (SIGGRAPH) 2008*
145. Z. Fan, F. Qin, A.E. Kaufamm, S. Yoakum-Stover*, GPU cluster for Hgh Performance Computing in : Proceedings of ACM/IEEE Superocmputing Conference 2004 pp. 47-59.*
146. J. Kriiger, R. Wetermann, *Linear Algeria operators for GPU implementation of Numerical Algorithms ACm Tran, Graph (SIGGRAPH) 22 (3) pp. 908-916. (2003)*
147. Tutorial SC 2007 SC05 : *High Performance Computing with CUDA*
148. *FASTRA* http://www.fastra.ua.ac.bc/en/faq.html
149. *AMD Stream Computing software Stack ;* http://www.amd.com
150. *BrookGPU :* http://graphics standafrod.edu/projects/brookgpu/index.html
151. *FFT – Fast Fourier Transform* www.fftw.org
152. BLAS – *Basic Linear Algebra Suborutines –* www.netlibr.org/blas
153. LAPACK : *Linear Algebra Package –* www.netlib.org/lapack
154. Dr. Larry Seller, Senipr Principal Engineer; Larrabee : A Many-core Intel Architecture for Visual computing, Intel Deverloper FORUM 2008
155. *Tom R Halfhill,* Intel's Larrabee Redefines GPUs – Fully Programmable Many core Processor Reaches Beyond Graphics, Microprocessor Report September 29, 2008
156. Tom R Halfhill AMD's Stream Becomes a River – Parallel Processing Platform for ATI GPUs Reaches More Systems, Microprocessor Report December 2008
157. AMD's ATI Stream Platform http://www.amd.com/stream
158. General-purpose computing on graphics processing units (GPGPU) http://en.wikipedia.org/wiki/GPGPU
159. Khronous Group, OpenGL 3, December 2008 URL : http://www.khronos.org/opencl

References & Acknowledgements

References

160. NVIDA CUDA C Programming Guide Version V4.0, May 2012 (5/6/2012) http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf
161. NVIDIA Developer Zone http://developer.nvidia.com/category/zone/cuda-zone
162. NVIDIA CUDA Toolkit 4.0 (May 2012) http://developer.nvidia.com/cuda-toolkit-4.0
163. NVIDIA CUDA Toolkit 4.0 Downloads http://developer.nvidia.com/cuda-toolkit
164. NVIDIA Developer ZONE – GPUDirect http://developer.nvidia.com/gpudirect
165. NVIDIA OpenCL Programming Guide for the CUDA Architecture version 4.0 Feb, 2012 (2/14,2012) http://developer.download.nvidia.com/compute/DevZone/docs/html/OpenCL/doc/OpenCL_Programming_Guide.pdf
166. Optimization : NVIDIA OpenCL Best Practices Guide Version 1.0 Feb 2012 http://developer.download.nvidia.com/compute/DevZone/docs/html/OpenCL/doc/OpenCL_Best_Practices_Guide.pdf
167. NVIDIA OpenCL JumpStart Guide - Technical Brief http://developer.download.nvidia.com/OpenCL/NVIDIA_OpenCL_JumpStart_Guide.pdf
168. NVIDA CUDA C BEST PRACTICES GUIDE (Design Guide) V4.0, May 2012
169. http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Best_Practices_Guide.pdf
170. NVIDA CUDA C Programming Guide Version V5.0, May 2012 (5/6/2012)
171. http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf
172. Programming Massively Parallel Processors - A Hands-on Approach, David B Kirk, Wen-mei W. Hwu, Nvidia corporation, 2010, Elsevier, Morgan Kaufmann Publishers, 2011
173. Aftab Munshi Benedict R Gaster, timothy F Mattson, James Fung, Dan Cinsburg, Addison Wesley, OpenCL Progrmamin Guide, Pearson Education, 2012
174. The OpenCL 1.2 Specification Khronos OpenCL Working Group
175. http://www.khronos.org/files/opencl-1-2-quick-reference-card.pdf" The OpenCL 1.2 Quick-reference-card ; Khronos OpenCL Working Group
176. http:/www.openmp.org OpenMP 4.0

Thank You

Any questions ?