OpenMP: An Overview

Multicore Processors

Agenda

- Motivation: The Need
- The OpenMP Solution
- OpenMP Features
- OpenMP Implementation
- Getting Started with OpenMP on KeyStone

Agenda

- **Motivation: The Need**
- The OpenMP Solution
- OpenMP Features
- OpenMP Implementation
- Getting Started with OpenMP on KeyStone

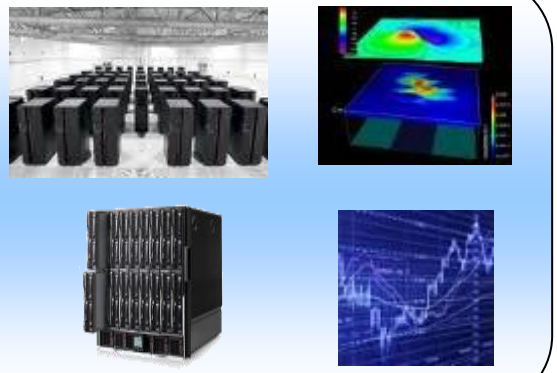
Motivation: TI Multicore Perspective

Mission Critical

Test and Automation

Medical Imaging

High Performance Compute



Emerging Broadband

Emerging

Multichannel & Next Generation Video – H.265, SVC, etc.

Motivation: Migrate SW from Single to Multicore

- **Earlier with Single Core**
  - New, faster processor would give desired performance boost
  - Faster execution speed was a result of better hardware
  - Minimal effort from software developers
  - Porting sequential code was straight forward

- **Now with Multicore**
  - Boost in performance not only function of hardware
  - Need to master software techniques that leverage inherent parallelism of multicore device
  - Every semiconductor vendor has own software solution
  - Many are new to multicore software development and have existing sequential code to port

Motivation: The Need

An efficient way to program multicore that is:

- Easy to use and quick to implement
- Scalable
- Sequential-coder friendly
- Portable and widely adopted

Agenda

- Motivation: The Need
- **The OpenMP Solution**
- OpenMP Features
- OpenMP Implementation
- Getting Started with OpenMP on KeyStone

The OpenMP Solution

What is OpenMP?

- An API for writing multi-threaded applications
- API includes compiler directives and library routines
- C/C++ and Fortran support in general. C/C++ support on TI devices.
- Standardizes last 20 years of Shared-Memory Programming (SMP) practice

The OpenMP Solution:

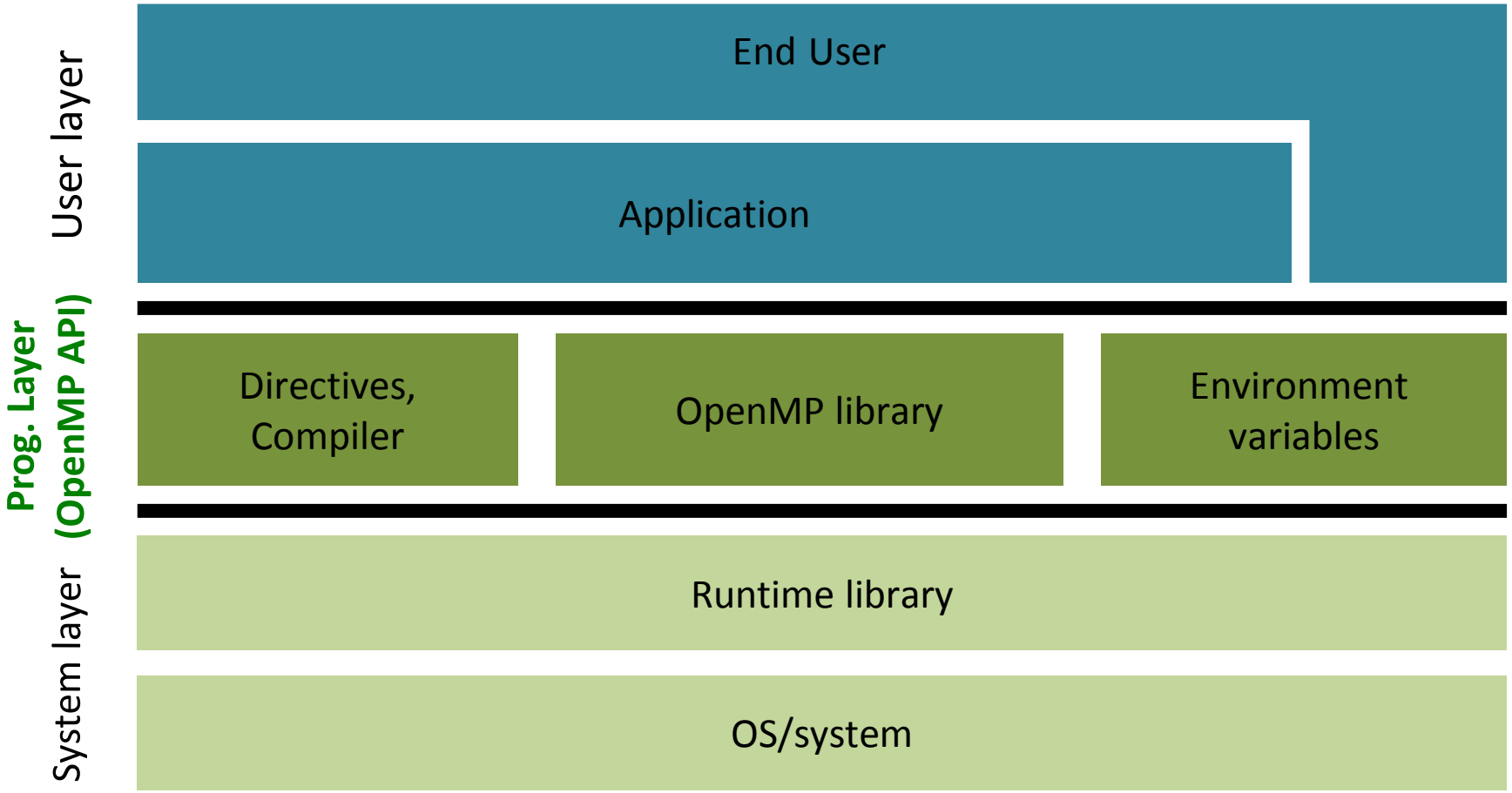
How does OpenMP address the needs?

- Easy to use and quick to implement
  - Minimal modification to source code
  - Compiler figures out details
- Scalable
  - Minimal or no code changes to add cores to implementation
- Sequential-coder friendly
  - Allows incremental parallelization v/s all-or-nothing approach
  - Allows unified code base for sequential and parallel versions
- Portable and widely adopted
  - Ideal for shared-memory parallel (SMP) architectures
  - Open-source and community-driven effort
  - Architecture Review Board includes:TI, Cray, Intel, NVidia, AMD, IBM, HP, Microsoft and others

Agenda

- Motivation: The Need
- The OpenMP Solution
- **OpenMP Features**
- OpenMP Implementation
- Getting Started with OpenMP on KeyStone

Features: OpenMP Solution Stack



Features: OpenMP API consists of…

- **Compiler Directives and Clauses:**
  - Specifies instructions to execute in parallel and its distribution across cores
  - Example: `#pragma omp construct [clause [clause] .. ]`

- **Library Routines:**
  - *Execution Environment Routines*
    - Configure and monitor threads, processors, and parallel environment
    - Example: `int omp_set_num_threads (int)`
  - *Lock Routines*
    - Synchronization with OpenMP locks
    - Example: `void omp_set_lock (omp_lock_t *)`
  - *Timing Routines*
    - Support portable wall clock timer
    - Example: `double omp_get_wtime(void)`

- **Environment Variables:**
  - Alter execution features of applications like default number of threads, loop iteration scheduling, etc.
  - Example: `OMP_NUM_THREADS`

Agenda

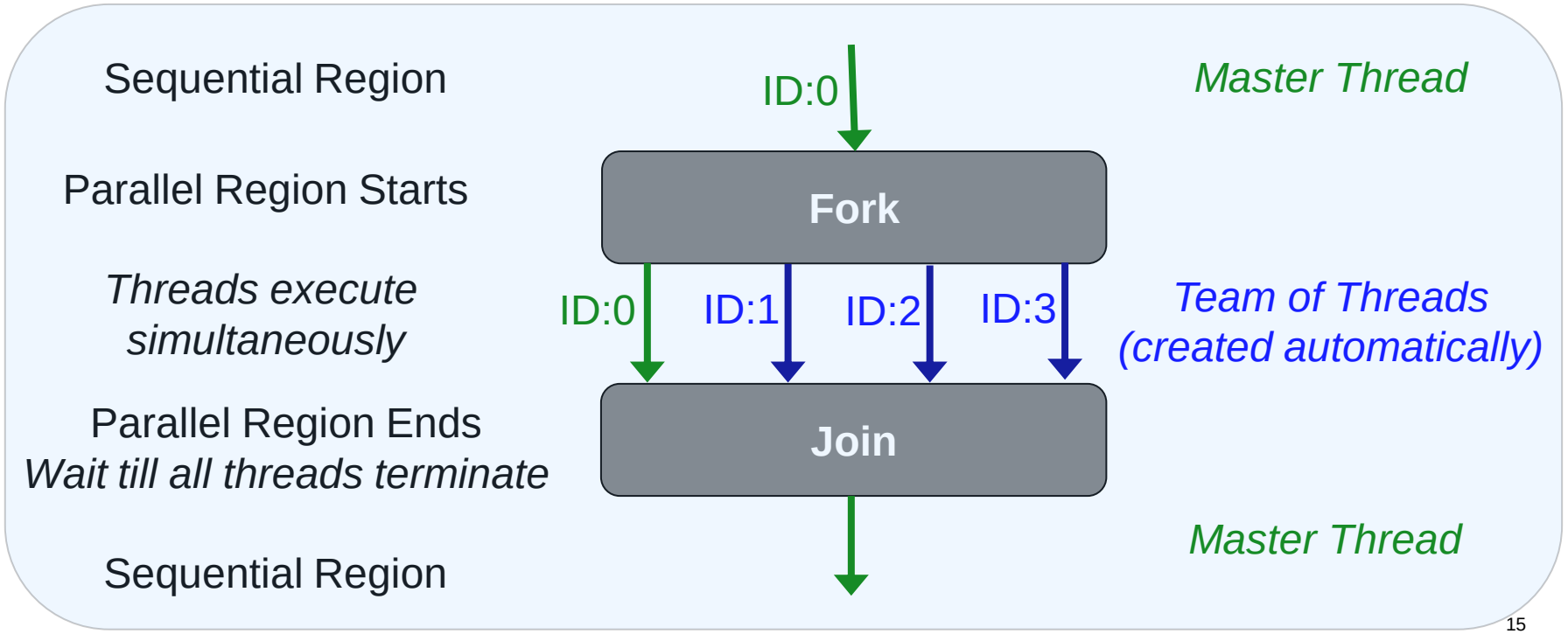
- Motivation: The Need
- The OpenMP Solution
- OpenMP Features
- **OpenMP Implementation**
  - **Create Teams of Threads**
  - **Share Work among Threads**
  - **Manage Data-Scoping**
  - **Synchronize Threads and Variables**
- Getting Started with OpenMP on KeyStone

Agenda

- Motivation: The Need
- The OpenMP Solution
- OpenMP Features
- **OpenMP Implementation**
  - **Create Teams of Threads**
  - Share Work among Threads
  - Manage Data-Scoping
  - Synchronize Threads and Variables
- Getting Started with OpenMP on KeyStone

Implementation: Use OpenMP to…

- **Create Teams of Threads**
  - Fork-Join Model
  - Execute code in a parallel region
  - Implemented by using compiler directive `#pragma omp parallel`
  - Nesting 'parallel' directives is possible, allowing multilevel parallelism



Implementation: Parallel Construct

```
#include <ti/omp/omp.h>
void main()
{
    omp_set_num_threads(4);

    #pragma omp parallel
    {
        int tid = omp_get_thread_num();

        printf ("Hello World from thread = %d\n", tid);

    }
}
```

Include Header
API definitions

Library Function
Set # of threads
(typically # of cores)

Compiler Directive
Fork team of threads

Library Function
Get thread id

Implicit Barrier

Agenda

- Motivation: The Need
- The OpenMP Solution
- OpenMP Features
- **OpenMP Implementation**
  - Create Teams of Threads
  - **Share Work among Threads**
  - Manage Data-Scoping
  - Synchronize Threads and Variables
- Getting Started with OpenMP on KeyStone

Implementation: Use OpenMP to…

- **Share Work among Threads**
  - By default each thread redundantly executes all code in // region
  - Programmer can insert ***work-sharing constructs*** to express how computation should be distributed

  - Example: Distribute *for* loop
    - Applicable only to loops where iterations are independent, i.e. changing order of execution won't matter
    - `#pragma omp for`

  - Example: Distribute multiple tasks
    - `#pragma omp section`

Implementation: Work-sharing Constructs

Sequential Code

```
for(i=0;i<N;i++)   { a[i] = a[i] + b[i]; }
```

Only with Parallel Construct

```
#pragma omp parallel
{
      int id, i, Nthrds, istart, iend;
      id = omp_get_thread_num();
      Nthrds = omp_get_num_threads();
      istart = id * N / Nthrds;
      iend = (id+1) * N / Nthrds;
      for(i=istart;i<iend;i++)   { a[i] = a[i] + b[i]; }
}
```

Parallel and Work-sharing Constructs

```
#pragma omp parallel
#pragma omp for
      for(i=0;i<N;i++)   { a[i] = a[i] + b[i]; }
```


Source: Reference #3

Implementation: Work-sharing Constructs

```
#pragma omp parallel
#pragma omp sections
{
    #pragma omp section
        x_calculation();

    #pragma omp section
        y_calculation();

    #pragma omp section
        z_calculation();
}
```

By default, there is a barrier at the end of the “omp sections”
Use the “nowait” clause to turn off the barrier.

Source: Reference #5

Agenda

- Motivation: The Need
- The OpenMP Solution
- OpenMP Features
- **OpenMP Implementation**
  - Create Teams of Threads
  - Share Work among Threads
  - **Manage Data-Scoping**
  - Synchronize Threads and Variables
- Getting Started with OpenMP on KeyStone

Implementation: Use OpenMP to…

- **Manage Data-scoping using Clauses**
  - Control how variables should be treated in a parallel region
  - Clauses
    - ***private*** clause
      - Each thread has a private copy of this variable and a unique value throughout the parallel construct
      - Variable declared inside parallel region is automatically private
      - Stored in thread stack; default size set by compiler but can override
    - ***shared*** clause
      - Same copy of this variable is seen by all threads
      - Variable declared outside parallel region is automatically shared (part of MSMC or DDR3)
    - ***default*** clause
      - Override default scope assigned to any variable
      - Set to ***none*** to explicitly specify scope of all variables used inside //
  - Programmer's responsibility to declare which variables are shared / private
  - Some variables like iteration counts, the compiler automatically enforces

Implementation: Data-Scoping Clauses

```
#pragma omp parallel for default (none) private( i, j, sum )
shared (A, B, C) if (flag)
{
    for (i = 0, i < 10; i++) {
        sum = 0;
        for ( j = 0; j < 20; j++ )
            sum += B[ i ][ j ] * C [ j ];
        A[ i ] = sum;
    }
}
```

Agenda

- Motivation: The Need
- The OpenMP Solution
- OpenMP Features
- **OpenMP Implementation**
  - Create Teams of Threads
  - Share Work among Threads
  - Manage Data-Scoping
  - **Synchronize Threads and Variables**
- Getting Started with OpenMP on KeyStone

Implementation: Use OpenMP to…

- **Synchronize Threads**
  - Synchronization at the end of work-sharing or // construct is automatic.
  - Synchronizing subset of threads has to be manually handled.
  - Some Synchronization Directives:
    - **`#pragma omp critical <name>`**
      - Only one thread may enter at a time.
      - Applies to block of code
      - If critical sections are unnamed, threads will not enter any of them.
    - **`#pragma omp atomic`**
      - Hardware provides atomic operation for expression
      - Applies to line of code (expression like X+=5)
      - Less overhead but less portability and limited to specific operations
    - **`#pragma omp barrier`**
      - Each thread waits until all threads arrive
    - **`#pragma omp flush[optional list]`**
      - User can creates sequence point for consistent view of memory
      - Implicit barriers automatically ensure cache coherency

Implementation: Synchronization Constructs

```
int sum = 0, i;

int A [100] = populate();

#pragma omp for shared (sum, A)
 {
     for (i = 0, i < 100; i++) {

     #pragma omp atomic
                sum += A [ i ];

     }

 }
```

Implementation: Reduction Construct

```
int sum = 0, i;

int A [100] = populate();

#pragma omp for shared (A) reduction (+:sum)
 {
     for (i = 0, i < 100; i++) {

                sum += A [ i ];

     }
 }
```

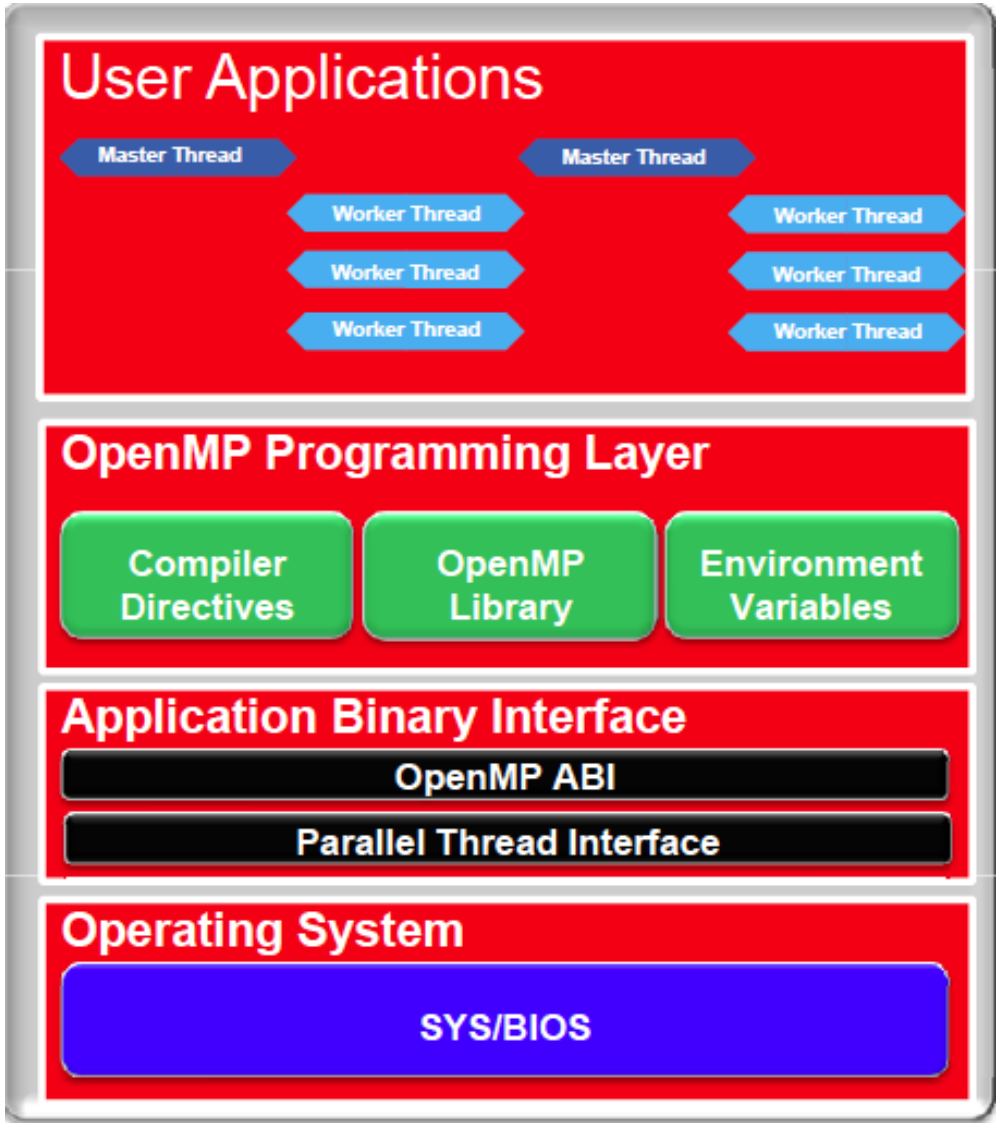
- Reduction creates private copy of shared variable for each thread
- At end of parallel loop, private copies of variable are ‘reduced’ back into original shared variable and operator (‘+’) is applied

Agenda

- Motivation: The Need
- The Solution: OpenMP
- OpenMP Features
- OpenMP Implementation
- **Getting Started with OpenMP on KeyStone**

OpenMP on KeyStone: Solution Stack

- Depends on SYS/BIOS. Each core runs the RTOS.
- OpenMP master and worker threads execute inside dedicated SYS/BIOS tasks.
- IPC is used for communication and synchronization.
- OpenMP run-time state and user data is allocated in shared memory.



Source: Reference #3

OpenMP on KeyStone: Availability

- TI's OpenMP package 'OMP' is installed as part of the MCSDK installation w/ OpenMP programming layer and runtime, and CodeGen 7.4.x compiler.
- MCSDK v2.1.2.6 is the latest version available, and includes OMP v1.1.3.2
- Download MCSDK at http://software-dl.ti.com/sdoemb/sdoemb_public_sw/bios_mcsdk/latest/index_FDS.html
- Once MCSDK is installed, install update patch for OMP v1.02.00 from http://software-dl.ti.com/sdoemb/sdoemb_public_sw/omp/1_02_00_05/index_FDS.html

OpenMP on KeyStone: Recent Updates

Important to update to latest OMP version which includes:

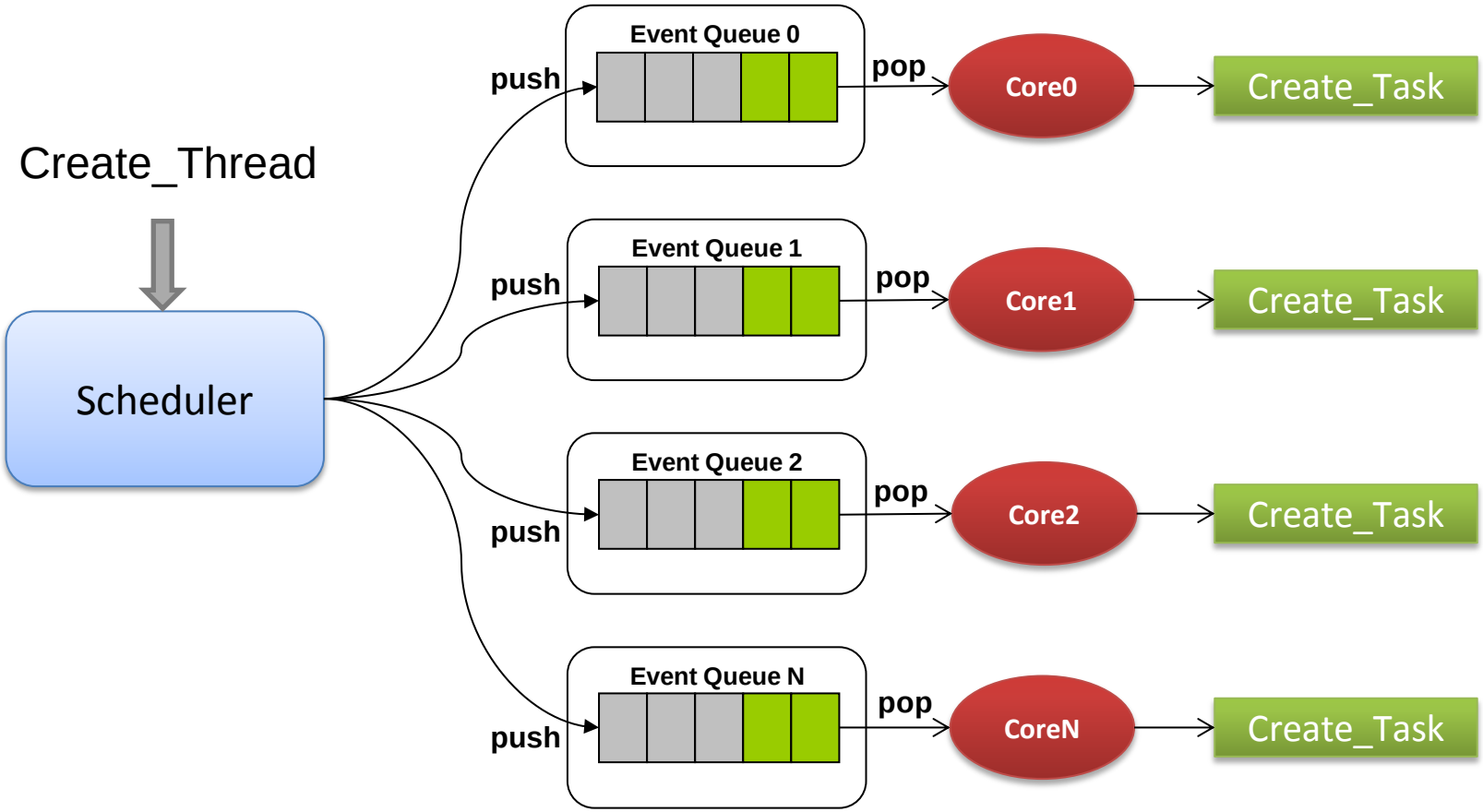
- Support for TI's C6657 device, and nested parallelism feature
- Significant performance boost from previous versions: Inter-core synchronization achieved through Multicore Navigator instead of hardware semaphores, and other optimizations
- BIOS/IPC related memory for each core moved to MSMC, making L2 entirely available to the developer
- TIP: Assign OpenMP variable heap & .data memory to MSMC/DDR, and not L2, to ensure variable sharing across cores
- Refer to OMP release notes and user guide for more details on improvements, bug fixes, and configuration tips related to this release

OpenMP on KeyStone: What is Coming

- The next OMP update will incorporate the OpenMP 3.1 specification.
- Support for TI's KeyStone II devices is also anticipated in the next release.
- The OpenMP 3.2 specification, which is expected from the OpenMP Architecture Review Board later this year, will incorporate heterogeneous multicore support. This specification will then be incorporated into TI's OMP package to support ARM+DSP devices.

OpenMP on KeyStone: Spawning Threads

- Use of an event queue for each core for task assignments
- Scheduler keeps track of the number of threads per core to distribute threads evenly on the cores.



Source: Reference #3

OpenMP on KeyStone: CCS Demo

We will see how to:

- Access example OpenMP projects from CCS v5.1
- Include OpenMP header file

```
#include <ti/omp/omp.h>
```

- Specify number of cores in project configuration .cfg

```
OpenMP.setNumProcessors(4);
```

- Provide *--omp* compiler option…available as a check box in project settings on CCSv5.1

```
Build → C6000 Compiler → Advanced Options
→ Advanced Optimizations → Enable Support
for OpenMP 3.0
```

References

1. *OpenMP on C6000 wiki article*
   http://processors.wiki.ti.com/index.php/OpenMP_on_C6000

2. *MCSDK User Guide*
   http://processors.wiki.ti.com/index.php/BIOS_MCSDK_2.0_User_Guide

3. *OpenMP Runtime for SYS/BIOS User's Guide*
   Included in OMP/docs folder once you install OMP package

4. *OpenMP Specification,* http://openmp.org/wp/openmp-specifications/

5. *Using OpenMP*, B. Chapman, G. Jost, R. Van Der Pas
   http://www.amazon.com/Using-OpenMP-Programming-Engineering-Computation/dp/0262533022/

6. *Introduction to OpenMP*
   http://community.topcoder.com/tc?module=Static&d1=features&d2=091106