OpenMP

Arash Bakhtiari
bakhtiar@in.tum.de

2012-12-18 Tue

Introduction

- Chip manufacturers are rapidly moving to multi-core CPUs

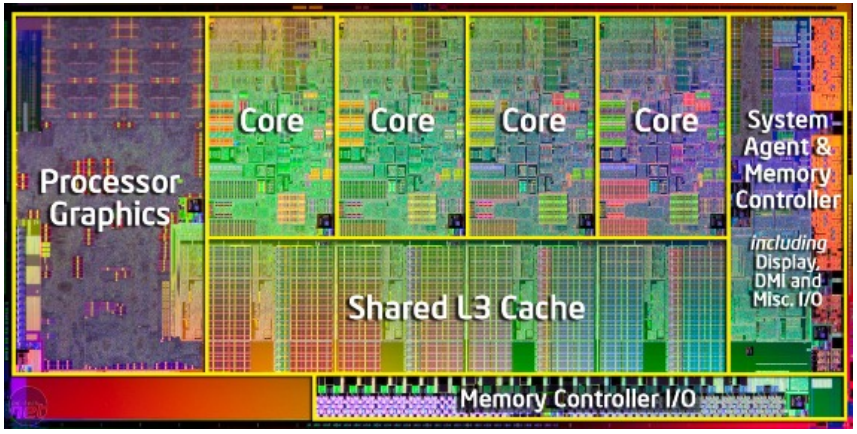

Figure : Quad-core processor Intel Sandy Bridge

Shared Memory Model

- All processors can access all memory in global address space.
- Threads Model: A single process can have multiple, concurrent execution paths
- On a multi-core system, the threads run at the same time, with each core running a particular thread or task.

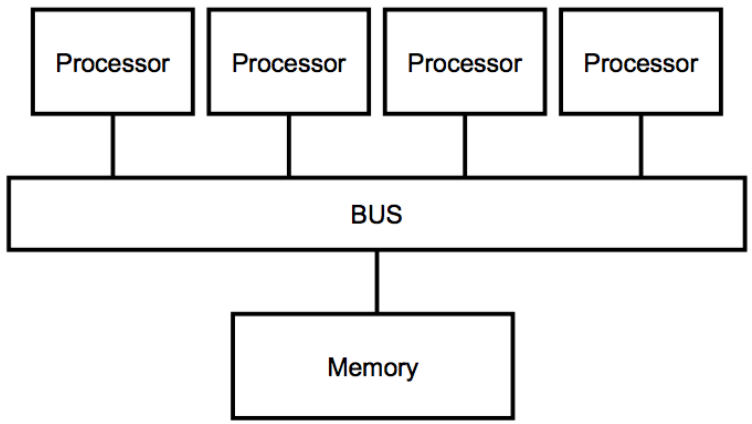

Figure : Shared Memory Model [1]

What is OpenMP?

- An Application Program Interface (API)
- Used to explicitly direct multi-threaded, shared memory parallelism
- Provides a portable, scalable model
- Supports C/C++ and Fortran on a wide variety of architectures

Fork-Join Model

- OpenMP-program starts as a single thread
- Additional threads (Team) are created when the master hits a parallel region
- When all threads finished the parallel region, the new threads are given back to the runtime or operating system.
- The master continues after the parallel region

Fork-Join Model (cont.)

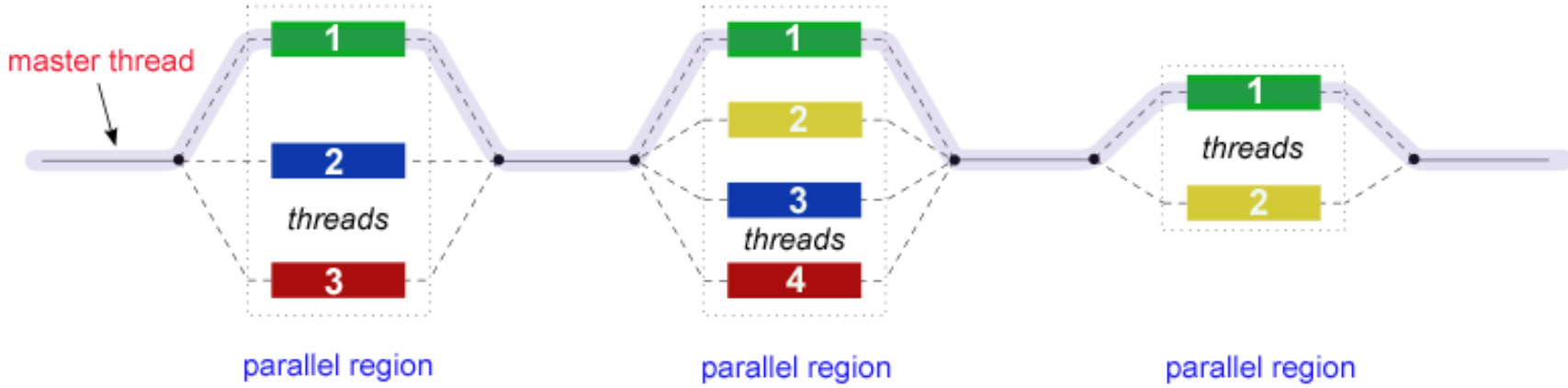

Figure : Fork-Join Model [1]

OpenMP API

Primary API components:

- Compiler Directives:

```
#pragma omp parallel
```

- Run-time Library Routines:

```
int omp_get_num_threads(void);
```

- Environment Variables

```
export OMP_NUM_THREADS=2
```

Example

Listing 1: OpenMP Hello World!

```
#include <iostream>
#include <omp.h>

int main(int argc, char *argv[])
{

#pragma omp parallel
  {
    std::cout << "THREAD: " << omp_get_thread_num() << "\tHello, World!\n";
  }
  return 0;
}
```

Listing 2: Compiling

```
g++ -o hello hello.c -fopenmp
```

Classification of Variables

- private(var-list):
  - Variables in var-list are private
- shared(var-list):
  - Variables in var-list are shared.
- default(private | shared | none):
  - Sets the default for all variables in this region.

Example

Listing 3: OpenMP Private Variable

```
#include <iostream>
#include <omp.h>
int main(int argc, char *argv[])
{
  int i, j;
  i = 1;
  j = 2;
 std::cout << "BEFORE: i,j= "<< i << ", " << j << std::endl;

 #pragma omp parallel private(i)
  {
    i = 3;
    j = 5;
    std::cout << "IN-LOOP: i,j= "<< i << ", " << j << std::endl;
  }

  std::cout << "AFTER: i,j= "<< i << ", " << j << std::endl;
  return 0;
}
```

Work-Sharing Constructs

- Work-sharing constructs distribute the specified work to all threads within the current team
- Types:
  - Parallel loop
  - Parallel section
  - Master region
  - Single region

Parallel Loop

- Syntax:

```
#pragma omp for [clause ...]
```

- The iterations of the loop are distributed to the threads
- The scheduling of loop iterations: static, dynamic, guided, and runtime.

Scheduling Strategies

- Schedule clause:

```
schedule (type [,size])
```

- **static**: Chunks of the specified size are assigned in a round- robin fashion to the threads.
- **dynamic**: The iterations are broken into chunks of the specified size. When a thread finishes the execution of a chunk, the next chunk is assigned to that thread.
- **guided**: Similar to dynamic, but the size of the chunks is exponentially decreasing. The size parameter specifies the smallest chunk. The initial chunk is implementation dependent.
- **runtime**: The scheduling type and the chunk size is determined via environment variables.

Example

Listing 4: OpenMP Private Variable

```
#include <iostream>
#include <omp.h>

#define CHUNKSIZE 100
#define N     1000
int main ()
{
  int i, chunk;
  double a[N], b[N], c[N];
  srand ( time(NULL) );
  for (i=0; i < N; i++) {
    a[i] = generate_random_double(0.0, 10.0);
    b[i] = generate_random_double(0.0, 10.0);
  }
  chunk = CHUNKSIZE;

#pragma omp parallel shared(a,b,c,chunk) private(i)
  {

#pragma omp for schedule(dynamic,chunk) nowait
    for (i=0; i < N; i++)
      c[i] = a[i] + b[i];

  }
  return 0;
}
```

References

Blaise Barney, Lawrence Livermore National Laboratory, https://computing.llnl.gov/tutorials/openMP/