Intel TBB

11.01.2013

Concurrent execution models

- User level
  - Multiprogramming, time sharing
- Process level
  - Multitasking
- Thread level
  - Multi-threading

Processes

- PID
- Own
  - Memory address space
  - File handles
  - HW handles
- Share
  - Nothing
- Heavy
- Preemptive multi-tasked

Threads

- Exist within process
- Own
  - Stack
  - Copy of the registers
- Share
  - Memory
  - File handles
  - HW handles
- Light
- Preemptive multi-tasked

Intel TBB (Thread Building Blocks)

- C++ library
- Compatible with other threading packages
- Highly scalable
- Focus on threads
- Generic programming
- Work stealing task scheduler
- Open source [1]

Library contents

- Basic algorithms:
  `parallel_for, parallel_reduce, ...`
- Advanced algorithms:
  `parallel_pipeline, parallel_sort, ...`
- Containers:
  `concurrent_queue, concurrent_vector, ...`

Library contents

- Scalable memory allocation:
  `scalable_malloc, scalable_free, ...`
- Mutual exclusion:
  `mutex, spin_mutex, ...`
- Atomic operations:
  `fetch_and_add, compare_and_swap, ...`

Parallel_for example

- Implement the operation
  - C = A^2 + B^2
  - A, B, C arrays

Parallel_for example

- Create a function object
  - Object that performs only one operation
  - Operation implemented in overloading of ()

```
class vector_mult{
    double *const v1, *const v2;    // multiply these vectors
    double *v3;                     // put the result here
    public:
    // constructor copies the arguments into local storage
    vector_mult(double *vec1, double *vec2, double *vec3)
        : v1(vec1), v2(vec2), v3(vec3) { }
    // overload () so it does a vector multiply
    void operator() (const blocked_range<int> &r) const {
        for(int i=r.begin(); i!=r.end(); i++)
            v3[i] = v1[i] * v1[i] + v2[i] * v2[i];
    }
};
```

Parallel_for example

- Initialise the scheduler

```
task_scheduler_init init(num_of_tasks);
```

- `num_of_tasks`: optional, if not specified the library will choose the optimal number

Parallel_for example

- Launch the parallel for

```
parallel_for( blocked_range<int>(0, n, grain_size),
              vector_mult(v1,v2,v3) );
```

- Iteration space `(0, n)` broken into chunks each run in a separate thread
- `blocked_range<T>`: recursively divisible struct
- `grain_size`: optional, specifies the ideal size of a chunk
- `operator()` in `vector_mult` processes a chunk

Pthread, OpenMP, TBB

Challenges for parallel programming	Pthreads	OpenMP	TBB
Task level		x	x
Cross-platform support		x	x
Scalable runtime libraries			x
Threads' Control	x		
Pre-tested and validated		x	x
C Development support	x	x	
Intel® Threading Tools support	x	x	x
Maintenance for tomorrow		x	x
Scalable memory allocator			x
"light" mutex			x
Processor affinity	x		Thread affinity

Source: [2]

Thank You for the Attention

Questions are welcome

References

[1]:http://threadingbuildingblocks.org/

[2]:http://software.intel.com/en-us/blogs/2008/12/16/compare-windows-threads-openmp-intel-threading-building-blocks-for-parallel-programming/