Intel Thread Building Blocks

SPD course 2010-11

Massimo Coppola

30/05/2011

History

- Thread Building Blocks
- A library to simplify writing thread parallel programs and debugging them
- Originated circa 2006 as a commercial product
- Nowadays double licensed
  - Commercial version for industrial users
  - Open source version under GPLv2
    - Stable versions aligned with commercial ones
    - Developer, source-only versions
  - Multi-OS
    - Windows*, Linux, OS X direct support
    - Other o.s. support in the open source

What is TBB today

- A runtime + template library for C++
- Eases writing thread programs by raising the abstraction level
- Templates and classes are defined for
  - Common thread parallelism forms
  - Data structures to pass to parallel "skeletons"
  - Data structures to control parallelism
    - e.g. ranges
  - Operators to specify each skeleton semantics
- Runtime support library
  - Provide o.s. independent basic primitives
    - E.g. library task scheduling to threads, memory allocation

Supported abstractions

We will not see all… but feel free to study!

- **parallel_for**
- **lambda expressions**
- **parallel_reduce**
- parallel_do
- pipeline
  - Extended to dags as supersets of pipelines
- **concurrency-safe containers**
- **mutex helper objects**
- atomic<t> template (atomic operations)

Parallel for

- Express independent task computations
  - parallel_for (iteration space , function)
- Exploit a blocked_range template to express iteration space
  - 1D and 2D blocked ranges in the library
  - 3D version under development, o.s. available
- Automatic dispatch to independent threads
  - Heuristics within the library
  - Can be customized
    - Specify optional *partitioner* function to the parallel_for
    - Specify *grainsize* parameter in the range

Scheduling tasks to threads

- Task scheduler
  - Automatically created by the library
  - Customizable by program to suit user needs
    - Define scheduler creation/destruction time
    - Number of created threads
    - Stack size for threads
  - Customizable per construct
    - via construct parameters
- Much more in the docs about the scheduler
  - Task scheduler deals with pipelines and workflows

Choosing grain size

- As always, small grain size → high overhead
  - Intel suggests 100.000 clock cycles as grain size
  - Also suggests experimental procedure to set
  - You are expected to know already the issues, and take into account the number of cores and load balancing issues in your algorithm
- Cache affinity can impact performance
  - *affinity partitioner* tries to exploit it when scheduling tasks to threads

Type	Use	Conditions
simple	Chunks given by grain size (Default until TBB 2.2)	g/2 < chunk size <g
auto	Automatic size (heuristics, default nowadays)	g/2 < chunks size
affinity	Automatic size (heuristics to exploit affinity)	g/2 <chunksize

Lambda expression

- Upcoming in the next C++ standard
  - In latest candidate release of C++0x, April 2011
- Use a stereotype for in-place defining a free function (some support for storing the def)
  [variable_scope] function definition
- Capture variable references which are used inside, but defined outside the function
- Variable scope spec can dictate capturing by reference, by value, or disallow use
  - In general, e.g. [] disallow [=] by value [&] ref.
  - For specific variable(s)
    [=,&z] all by value, with only z by reference

Parallel reduce

- Expresses the reduction
  - parallel_reduce (iteration_space, function)
  - Iteration space defined as blocked_range
  - Function to apply
  - Function type differences w.r.t to parallel loop, op behaviour does not have the same const-requirements
  - Accepts an optional partitioner too

Container data Structures

- Data structures very often used in programs, whose thread-safe implementation is not trivial or does not match standard semantics
- Special care to avoid destroy program performance
- concurrent_hash_map
  - Constant or update access to elements
  - Access to elements can block other threads

Container data Structures

- concurrent_vector
  - Can grow as normal vector
  - Does not move its elements in memory
  - Destroying elements is not thread safe
- concurrent_queue
  - Simultaneous push/pop from concurrent threads
  - Ensure serialization and preserve object order
    - Bottleneck if improperly used
  - pop / push / try_push / size

Mutexes

- Classes to build *lock objects*
- The new lock object will generally
  - Wait according to specific semantics for locking
  - Lock the object
  - Release lock when destroyed
- Several characteristics of mutexes
  - Scalable
  - Fair
  - Recursive
  - Yeld / Block
- Check implementations in the docs:
  - mutex, recursive_mutex, spin_mutex, queueing_mutex, spin_rw_mutex, queueing_rw_mutex, null_mutex, null_rw_mutex
  - Specific reader/writer locks
  - Upgrade/downgrade operation to change r/w role

References

- Download docs and code from http://threadingbuildingblocks.org/
- Check the tutorial and reference