DEVELOPING SCALABLE APPLICATIONS
with Microsoft’s C++ Concurrency Runtime

Dana Groff, Microsoft
Senior Program Manager
DEVELOPING SCALABLE APPLICATIONS WITH MICROSOFT’S C++ CONCURRENCY RUNTIME
What we are talking about…

- This is a C++ talk

- I will not talk about C++ AMP in this session
  - Go to Daniel Moth’s session at 2pm in 406 titled “Blazing-fast code using GPUs and more, with Microsoft Visual C++”

- The focus of this talk is on refactoring using:
  - Parallel Patterns Library (PPL)
  - Asynchronous Agents Library (Agents)
  - Performance analysis with Visual Studio
  - And some debugging…
Map of C++ Concurrency

Programming Models

- Parallel Pattern Library
- Agents Library

Concurrency Runtime

- Task Scheduler
- Resource Manager

Tools

- Parallel Tasks
- Stacks
- Debugger
- Windows

Concurrent Visualizer

Data Structures

- Windows
- Visualizer
Parallel Programming is Hard

Success hinges on:
- Experience
- Friends
- Luck

…and cool tools
PERFORMANCE: that’s what counts...

This is what you want...
PERFORMANCE: that’s what counts…

This is what you have…
Refactoring your code to scale

- Is it your code?
  - I need to understand what is going on.

- Where do you start?
  - I need tools to guide where I focus my time.

- How much improvement can I expect to achieve?
  - Without spending too much time.
  - …and (re)writing a lot of code.

Debugger

Performance Analyzer

PPL & Agents
REMEMBER AMDAHL

\[
\frac{1}{(1-P) + \frac{P}{S}}
\]

- Set realistic expectations
- Focus your time where it counts
DEMO: REFACTORING OCEANS

...A STUDY IN REFACTORING UNFAMILIAR CODE FOR SCALING
Lessons Learned

**KISS**
- Optimize Serial Code First
- Optimize your loop size on serial code

**One step at a time**
- Add parallel code one piece at a time
- Focus on the high value code first
- What is high value changes for each run

**Fill in the gaps**
- Add asynchronous code to leverage both CPU and GPU at the same time.

**Don’t cut corners**
- Sloppy coding even when working with spaghetti will come to haunt you
- Never ever use [&] in a lambda
What did you just see?

- Guided refactoring using Visual Studio Performance analysis
- Easy scaling without a lot of rewriting using PPL
- Understanding the concurrency using the concurrency visualizer and debugger
Agents and Data Flow Concepts

Agents
• Has a queryable lifetime state
• You can wait for an agent to finish
• You can cancel an agent

Data Flow
• Connect message blocks together
• Filters may be defined
• Control flow with joins and choices
Asynchronous Agents Library

Message Blocks for Storing Data
- `unbounded_buffer<T>`
- `overwrite_buffer<T>`
- `single_assignment<T>`

Agent base class
- Start, stop
- State management

Message Control flow
- Choice, coin
- `transformer<T,U>, call<T>`
- `link_target`

Send and receive
- co-operatively send & receive messages
- `asend, send`
- `receive, try_receive`
Parallel Pattern Library Concepts

**Task**
- a computation that may be internally decomposed into additional tasks.

**Task group**
- a collection of tasks that form a logical computation or sub-computation.

**Parallel Loops**
- STL style algorithms with support for cancellation and composition.

**Continuation**
- a task that runs after a task; allows for wait-free algorithms
## Parallel Pattern Library

### Parallel Algorithms
- `parallel_for`
- `parallel_for_each`
- `parallel_invoke`

### Task Parallelism
- `task_handle`
- `task_group`
- `structured_task_group`

### Concurrent Containers
- `concurrent_queue`
- `concurrent_vector`
- `combinable`

### Sample Pack
- `parallel_for_fixed`
- `parallel_transform`
- `parallel_reduce`
- `parallel_sort`
- `parallel_buffered_sort`
- `parallel_radixsort`

### Sample Pack
- `task<T>`
  - `continue_with<T>`
  - `when_all<T>`
  - `when_any<T>`
  - `operators &&, ||`

### Sample Pack
- `concurrent_unordered_map`
- `concurrent_unordered_multimap`
- `concurrent_unordered_set`
- `concurrent_unordered_multiset`
Parallel Pattern Library

Parallel Algorithms
- parallel_for
- parallel_for_each
- parallel_invoke

V.next
- partitioners
- parallel_transform
- parallel_reduce
- parallel_sort
- parallel_buffered_sort
- parallel_radixsort

Task Parallelism
- task_handle
- task_group
- structured_task_group

V.next
- task<T>
  - then<T>
  - when_all<T>
  - when_any<T>
  - operators &&, ||

Concurrent Containers
- concurrent_queue
- concurrent_vector
- combinable

V.next
- concurrent_unordered_map
- concurrent_unordered_multimap
- concurrent_unordered_set
- concurrent_unordered_multiset

Other Planned Improvements

- Control NUMA workloads through locality
- Control Process Affinity
- No blocking when passing asynchronous messages
- Lower runtime overhead on large core-count machines
Resources

- ConcRT Extras / Sample Pack
  - http://code.msdn.com/ConcRTExtras
- Native Concurrency Forum
  - http://social.msdn.microsoft.com/Forums/parallelcppnative/
- Native Concurrency Team Blog
  - http://blogs.msdn.com/nativeconcurrency
- C++ Team Blog
  - http://blogs.msdn.com/vcblog
- MSDN Concurrency
  - http://www.msdn.com/concurrency
Questions?

Programming Models

Parallel Pattern Library

Agents Library

Concurrency Runtime

Task Scheduler

Resource Manager

Data Structures

Tools

Parallel Tasks Stacks Debugger Windows

Concurrency Visualizer

Dana.Groff@Microsoft.Com
Disclaimer & Attribution

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. There is no obligation to update or otherwise correct or revise this information. However, we reserve the right to revise this information and to make changes from time to time to the content hereof without obligation to notify any person of such revisions or changes.

NO REPRESENTATIONS OR WARRANTIES ARE MADE WITH RESPECT TO THE CONTENTS HEREOF AND NO RESPONSIBILITY IS ASSUMED FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

ALL IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED. IN NO EVENT WILL ANY LIABILITY TO ANY PERSON BE INCURRED FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD, the AMD arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. All other names used in this presentation are for informational purposes only and may be trademarks of their respective owners.

The contents of this presentation were provided by individual(s) and/or company listed on the title page. The information and opinions presented in this presentation may not represent AMD’s positions, strategies or opinions. Unless explicitly stated, AMD is not responsible for the content herein and no endorsements are implied.