NO CONCURRENT COLLECTIONS TUTORIAL

The Eighth Annual Concurrent Collections Workshop

2016/09/27

Frank Schlimbach

Agenda

Isolation

Intel® CnC…

Graphs

Separation and Isolation in CnC

Semantics/ Dependences From **Tuning and Platform**

What and How From **Where and When**

Control From **Data**

Consumer From **Producer**

Controller/Caller From **Controlee/Calee**

Semantics/Tuning

Abstract CnC

Domain-spec vs tuning spec

Intel® CnC

- Collections vs Tuners
  - Combination decided at compile-time
- Communication selected at runtime

Semantics/Tuning

Abstract CnC

- No code
- Tuning spec
  - Grouping
  - Placement
  - …

Intel® CnC

- Steps define the how
- Collections define what
- C++ API brings them together
- Tuners define
  - Placement
  - influence timing
  - …

What and How / Where and When

Abstract CnC

- No explicit concept of time
- No concept of space
- Collections are arbitrators/proxies
  - Put and get

Intel® CnC

- Steps define the how
- Collections define what
- Tuners define placement and influence timing
- A step use only IO collections for communication (sender/receiver is unknown)

Control/Data

Abstract CnC

- Control is explicit
  - Special edges or
  - Dedicated control collections
- Data is separate
  - "if" something needs to be done can be elsewhere (in time and space) than its input data

Intel® CnC

- Explicit control collections
  - Separates controller/controlee

Consumer/Producer, Controller/Controlee

Abstract CnC

- Collections serve as arbitrator

Intel® CnC

- Steps only communicate with IO collections
- Other side is unknown/not important
- Space and time are also irrelevant
- Runtime coordinates

Agenda

Isolation

Intel® CnC…

Graphs

Intel® CnC

C++ (open source, Linux, Windows, Xeon, Phi™…)

- Good performance through TBB, MPI and C++ template optimization potential

Domain API - Tuner API

- Linked/combined at compile-time

Shared and distributed memory

- With tuning only required data is transferred

Debugging aids

- Tracing: printf and Intel® Trace Analyzer and Collector
- Detailed timing

Intel® CnC - Advanced

Tuning

- Placement
- Scheduling (avoid re-try, priority)
- GC
- Sequentialization
- Cancelation
- Range/set partitioning
- Storage type (vector, hashmap, DB)

Lower-level APIs

- Communicator
- Scheduler
- Distributor (future)

Agenda

Isolation

Intel® CnC…

Graphs

Graphs

Goal: compose components but minimize requirements

⇒ Maximize exposed parallelism

⇒ Maximize flexibility

⇒ Minimize visibility

⇒ Keep it simple

Let their semantic dependences be the only restriction on parallelism.

- no implicit (or even explicit) barriers

Stick with the dependence theme.

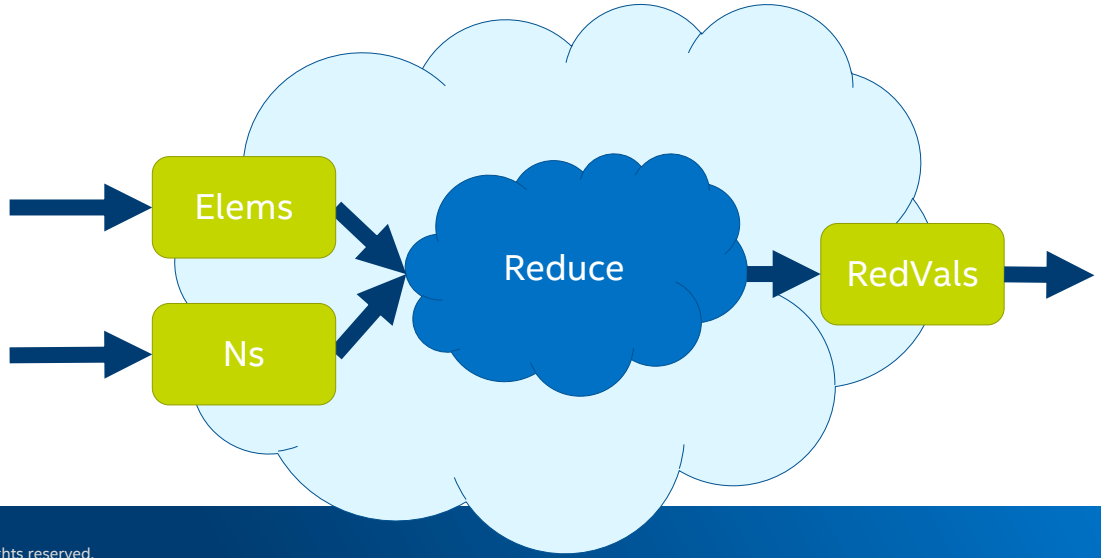
Resource utilization will be nicley isolated as a separate concern.

Example: Graph-like Reduction in Intel® CnC

- Operates on a “continuous” stream of incoming elements

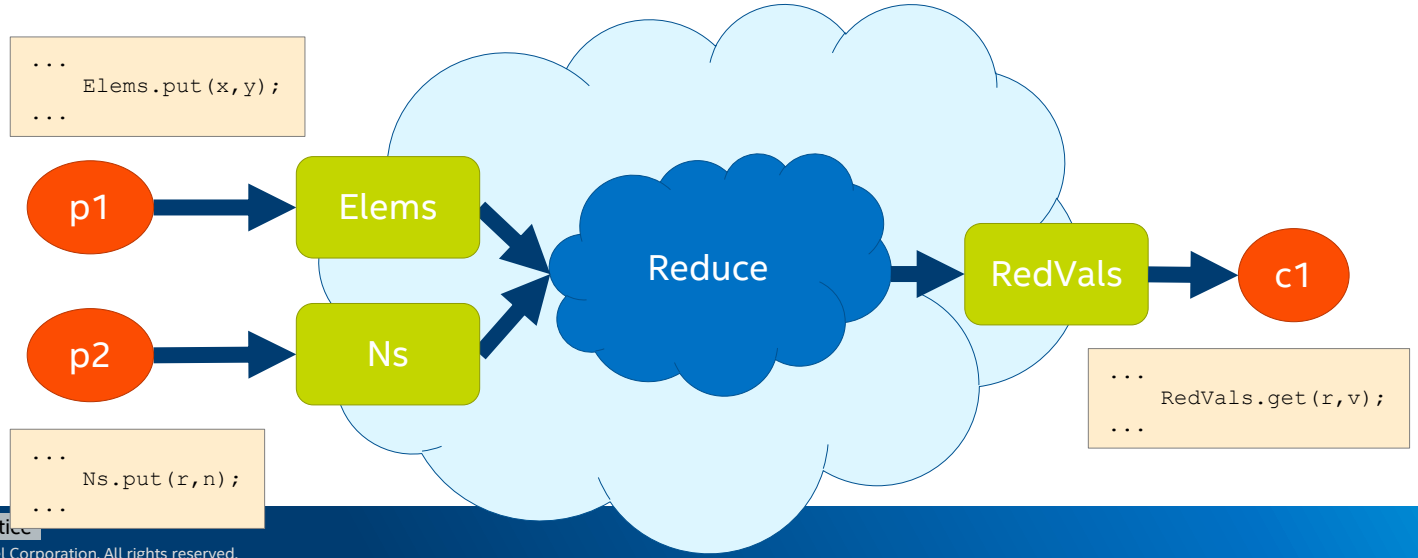
Example: Graph-like Reduction in Intel® CnC

- Operates on a “continuous” stream of incoming elements
- Continuous input accessed via callbacks
- Input/output to a graph handled through CnC collections



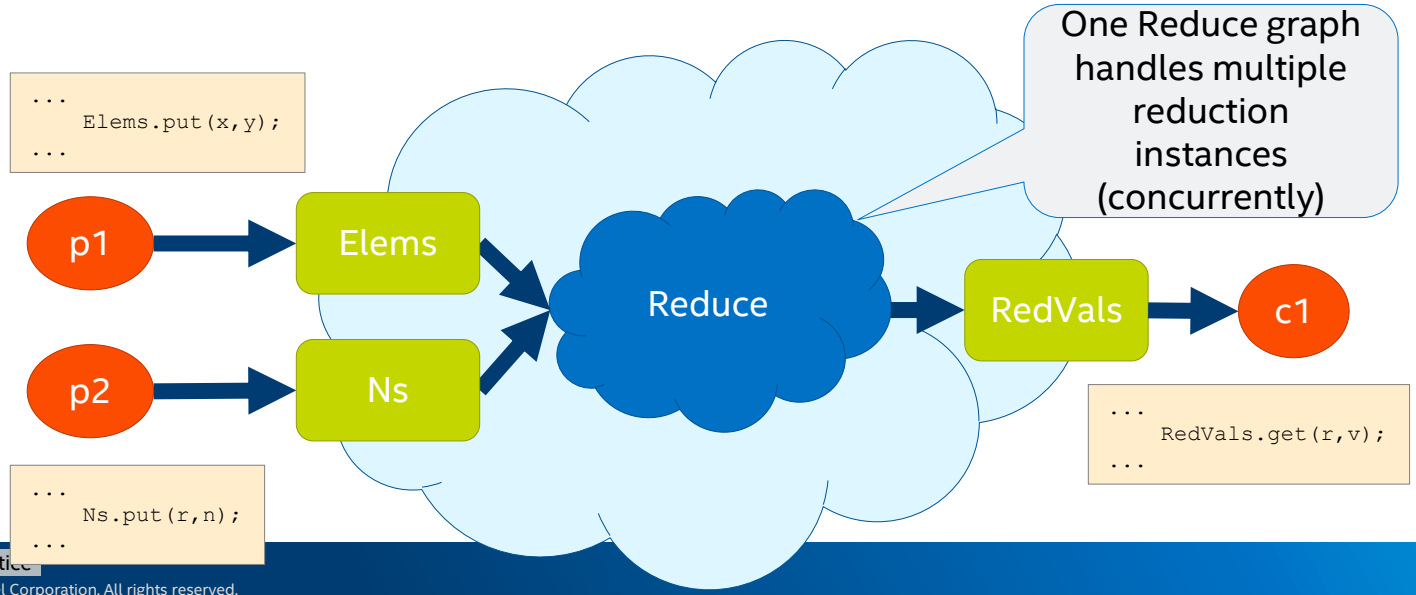
Example: Graph-like Reduction in Intel® CnC

- Operates on a “continuous” stream of incoming elements
- Continuous input accessed via callbacks
- Input/output to a graph handled through CnC collections
- Connected steps operate with usual put/get semantics



Example: Graph-like Reduction in Intel® CnC

- Operates on a “continuous” stream of incoming elements
- Continuous input accessed via callbacks
- Input/output to a graph handled through CnC collections
- Connected steps operate with usual put/get semantics



Example: Graph-like Reduction in Intel® CnC

- Operates on a “continuous” stream of incoming elements
- Continuous input accessed via callbacks
- Input/output to a graph handled through CnC collections
- Connected steps operate with usual put/get semantics

Ongoing Graph Discussions

Can a graph be prescribed and if so, what does it mean?

What's the "home" of IO collections?

Connecting runtimes through graphs?

Hierarchical data?

Hierarchical tag-spaces?

Legal Disclaimer & Optimization Notice

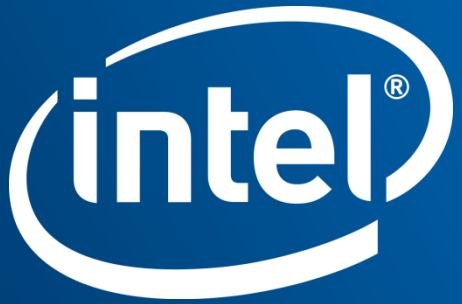
INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804


intel®