OVERWRITING, NON-DETERMINISTIC AND SAFE DATA-PUTS IN (INTEL®) CNC

The Eighth Annual Concurrent Collections Workshop

2016/09/27

Kath Knobe, Frank Schlimbach

Dynamic Single Assignment

Makes CnC

- Deterministic
- Safe
- Analyzable

Not always intuitive/suitable

- Sensors
- Multi-headed DBs
- Branch& Bound
- Re-using memory
- "GetAnyOne"

Current solution

Define a graph

- Internally allowed to be non-deterministic
- Graph can produce data and control
- Call-back interface intercepts unknown data tags

Not always trivial

Non-DSA items in CnC

Allow overwriting puts

- Non-default, explicit
- Per item(-collection)
- Non-deterministic
- Safe

Non-DSA items in CnC

Put(tag, value) Semantics

- Sets or updates "tag" to "value"
- Updates are atomic

Get(tag) Semantics

- Returns "value" for "tag" for some matching put(tag, value)
- Will not succeed without a put
- No other guarantees

Non-DSA items in Intel® CnC

Put(tag, value):

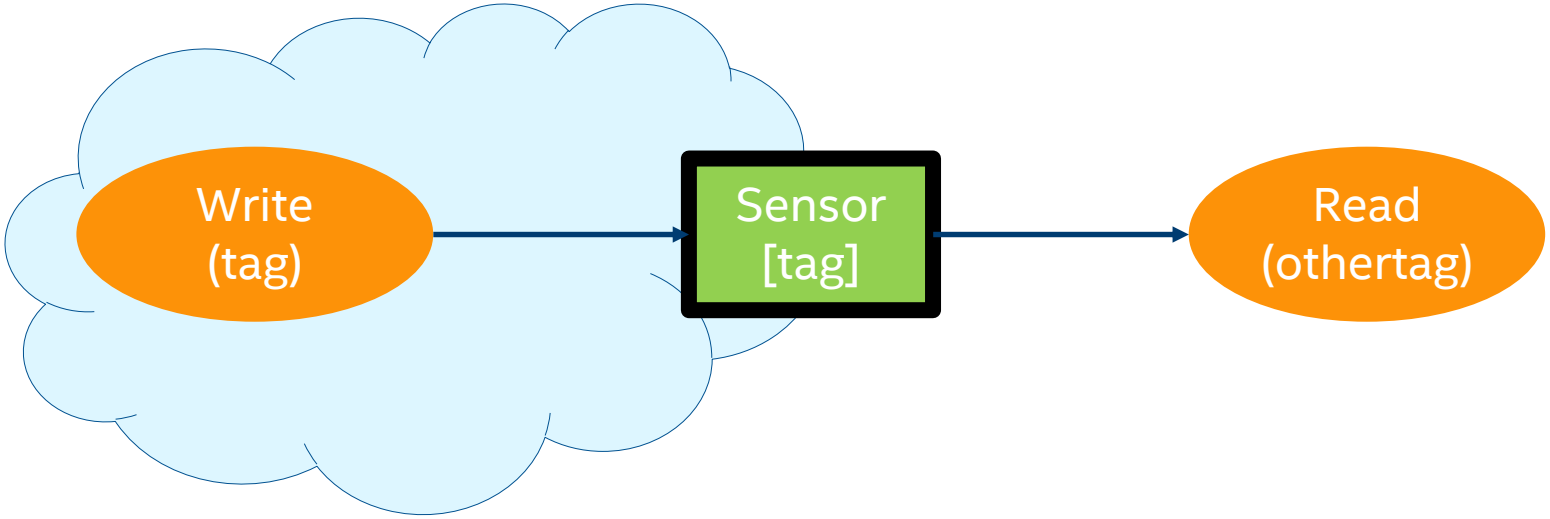
- Like a normal put
- Sets or updates "tag" to "value"
- Updates are atomic
- Updates "interested" remote copies
- On_put only executed at "first" put

Get(tag) Semantics

- Returns "value" for "tag" for some matching put(tag, value)
- Will not succeed without a put
- No other guarantees
- Gets the latest available value but does not check for updates

Example - sensor

Example - sensor



Example - sensor

```
/// producing sensor data
/// we get all latest input data and produce one combined data object
template< typename t1, typename t2, typename t3 >
int produce<t1, t2, t3 >::execute( const time_t, cv_context & ctxt ) const
{
    sensor1.get(0, v1);
    sensor2.get(0, v2);
    sonesor3.get(0, v3);
    ctxt.vals.put( 0, triple<t1, t2, t3 >( v1, v2, v3 ) );
    return 0;
}

/// dummy: just get the value tuples for given time (tag)
int consume::execute( const time_t, cv_context & ctxt ) const
{
    mytriple _vals;
    ctxt.vals.get( 0, _vals );
    CnC::Internal::Speaker oss;
    oss << _vals.val1 << " " << _vals.val2 << " " << _vals.val3 << " " ;
    return 0;
}
```

Example - sensor

```
// let our collections be non-DSA
struct ndsa_tuner : public CnC::hashmap_tuner
{
    template< typename Tag >
    bool is_dsa( const Tag & tag) const
    {
        return false;
    };
};
```

Per item!

Tuner is a template arg.
Constants get optimized out!

Overwriting (simplified)

```
acc = get_accessor(tag)
if not acc.insert_first (value) {
    if not tuner.isdsa(tag) {
        acc.update(value)
    } else ERROR
}
send_value_to_interested(tag, value)
```

Tuner is a template arg.
Constants get optimized out!

Advanced

Other cases require more than unconditional update

- Branch&bound

Extend overwrite-check with condition

In the works: conditional overwriting

We always have an old value!

```
// let our collections be DSA
struct ndsa_tuner : public CnC::hashmap_tuner
{
    template< typename Tag, typename Value >
    bool overwrite( const Tag & tag, const Value & old, const Value & new) const
    {
        return false;
    };
};
```

```
// overwrite only if new value is greater then the old one
struct ndsa_tuner : public CnC::hashmap_tuner
{
    template< typename Tag, typename Value >
    bool is_dsa( const Tag & tag, const Value & oldV, const Value & newV) const
    {
        return newV > oldV;
    };
};
```

Open questions

Should on_put be called for every put?

How to handle multiple puts on different processes?

- Currently ignored
- Re-evaluate the condition or not?

More examples!

More use cases!

Legal Disclaimer & Optimization Notice

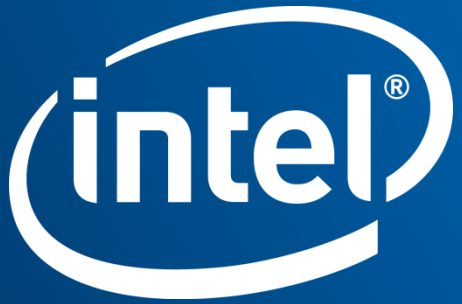
INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING 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®