TWO SMALL, EASY TO USE, OPENMP* FEATURES YOU MAY HAVE MISSED

Jim Cownie, Johnny Peyton
with help from Nitya Hariharan and Doug Jacobsen

Features We Discuss

Synchronization (lock) hints

The `nonmonotonic:dynamic` schedule

Both

- Were new in OpenMP 4.5
- May have slipped past you

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

SYNCHRONIZATION HINTS

Feature #1: Synchronisation Hints
What Are They?

A way to give more information to the OpenMP runtime about your use of locks, critical sections and atomics.

- In OpenMP 4.5 these were lock and critical section hints: `omp_lock_hint_*`
- In OpenMP 5.0 they will be generalised to include hints on atomics
  - Names change to `omp_sync_hint_*`
  - But the `omp_lock_hint_*` names are preserved (for now) in case you were already using them

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Why Do I Need Synchronisation Hints on Locks?

Different lock implementations have different performance properties

- Time to wake up on lock release
- Cost of contention (interference from waiting threads on working ones)
- Fairness

The LLLVM/Intel® runtime has **EIGHT** (!) possible lock implementations

- You can take a look at http://openmp.llvm.org

No one lock implementation can be optimal for all uses

The hints enable the use of advanced hardware features

- Intel® Transactional Synchronization Extensions (Intel® TSX)
- IBM*'s Hardware Transactional Memory in Power8* (and other) processors

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Hints on Locks and Critical

Hints are easy to apply

- Change `omp_init_lock` to `omp_init_lock_with_hint` and add the hint
- **No** changes to the places where the lock is used
- Apply hint to an `omp critical` statement
  - Must have a name
  - All criticals with the same name must have the same hint

Hints have no semantic implications

- Mutual exclusion is still guaranteed
- An implementation is allowed to completely ignore them!
  - But we hope it won't

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Hints on Locks and Critical

Express properties of the use of the lock/critical section

- `omp_sync_hint_contended`
  - We expect many threads to be waiting suggests use of a fair queueing lock
- `omp_sync_hint_uncontended`
  - We expect the lock normally to be free suggests use of an unfair spin-lock

Express suggested implementation

- `omp_sync_hint_speculative`
  - Use transactional synchronization hardware if it's available
- `omp_sync_hint_nonspeculative`
  - Don't use transactional synchronization hardware even if it's available

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Intel® Transactional Synchronization Extensions (Intel® TSX)

Can provide the performance of fine-grain reader/writer locks with many fewer code changes

Speculative execution:

- Hardware monitors read and write sets inside the transaction to detect conflicts and abort failing transactions
- Transactional writes are not seen by other threads until the transaction commits
- No forward progress guarantees => Need another strategy as well…

When using speculative locks from OpenMP that is all hidden in the library; you just see a normal lock

- Detectable using timestamps… (but that is a weird thing to do)

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Example: Wrapping `std::map`

Obvious approach to using `std::map` in an OpenMP code:

- Use a lock around all accesses

Note:

- Only changes to use hints are in red
- No changes to use of the lock

```
class lockedHash
{
    std::unordered_map<uint32_t, uint32_t> theMap;
    omp_lock_t theLock;
public:
    lockedHash(omp_sync_hint_t hint) {omp_init_lock_with_hint(&theLock,hint);}

    void insert(uint32_t key, uint32_t value) {
        omp_set_lock(&theLock);      // Claim the lock
        theMap.insert({key,value});
        omp_unset_lock(&theLock);    // Release the lock
    }

    uint32_t lookup(uint32_t key) {
        omp_set_lock(&theLock);      // Claim the lock
        auto result = theMap.find(key);
        omp_unset_lock(&theLock);    // Release the lock
        return result == theMap.end() ? 0 : result->second;
    }
};
```

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Performance of wrapped `std::map` with different hints

All threads are doing lookups or insertions on random elements of a 10,000 entry hash.

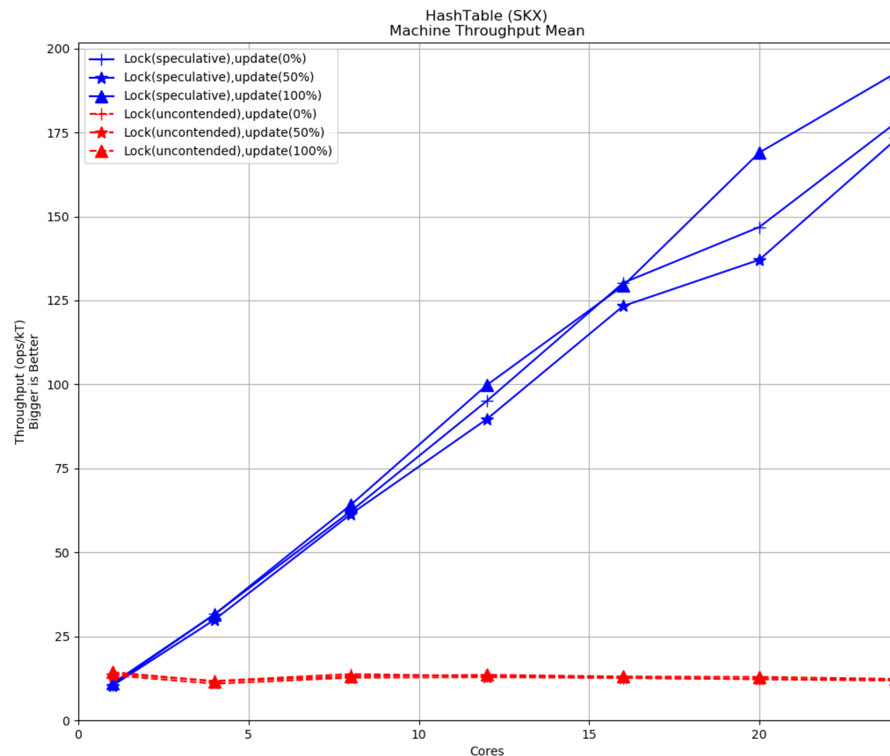
Measure three different proportions

- All insertions
- 50% of each
- All lookups

Machine is 24C Xeon® Platinum 8168 at 2.7GHz

Running 1Thread/core

The only change is the hint used



Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

What About Replacing Atomics With Locks?

These speculative locks seem great!

Could we use them to replace blocks of atomic operations?

Test case probability of conflict is **very** low

- Each thread picks random nupdates long element in an nupdates*1024*1024 float array

```
template <uint32_t nupdates>
static void doUpdateAtomicContig(float *values)
{
#pragma unroll(nupdates)
    for (int j=0; j<nupdates; j++)
#pragma omp atomic
        values[j] += 1.0;
}

template <uint32_t nupdates>
static void doUpdateLockContig(float * values)
{
    omp_set_lock(&criticalLock);
#pragma unroll(nupdates)
    for (int j=0; j<nupdates; j++)
        values[j] += 1.0;
    omp_unset_lock(&criticalLock);
}
```

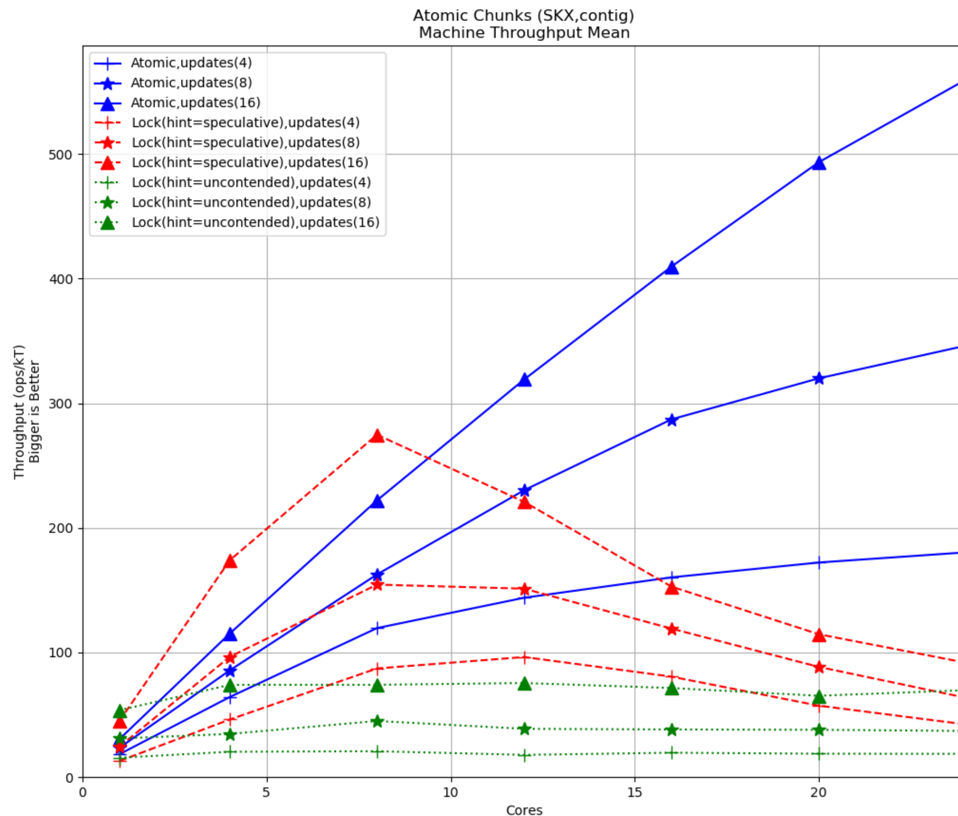
Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Atomic Replacement Performance

Atomics remain better even than the speculative locks

I did it so you don't have to!

Same machine as before (24C Xeon® Platinum 8168 at 2.7GHz)



Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Synchronisation Hint Conclusions

Synchronisation hints are not a panacea

BUT

- They are standard OpenMP, so all compilers should accept them (even if they then ignore them!)
- They are easy to use (usually a single line change)
- They can give significant performance improvement in some cases

If you use locks and critical sections consider providing a hint and try speculation.

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

THE `nonmonotonic:dynamic` SCHEDULE

schedule(nonmonotonic:dynamic) Semantics

The dynamic schedule has existed “forever”, what’s this “nonmonotonic” thing?

Consider the code on the right.

Can it abort?

Before OpenMP 5.0 that was unclear...

- OpenMP 4.5 added the monotonic and nonmonotonic qualifiers to schedules so that you can say what you need
- OpenMP 5.0 says that the unqualified dynamic schedule is nonmonotonic (so it can abort!)

```
#pragma omp parallel
{
    int myMax = 0;
#pragma omp for schedule(dynamic)
    for (int i=0; i<N; i++)
    {
        if (i<myMax) abort();
        myMax = i;
    }
}
```

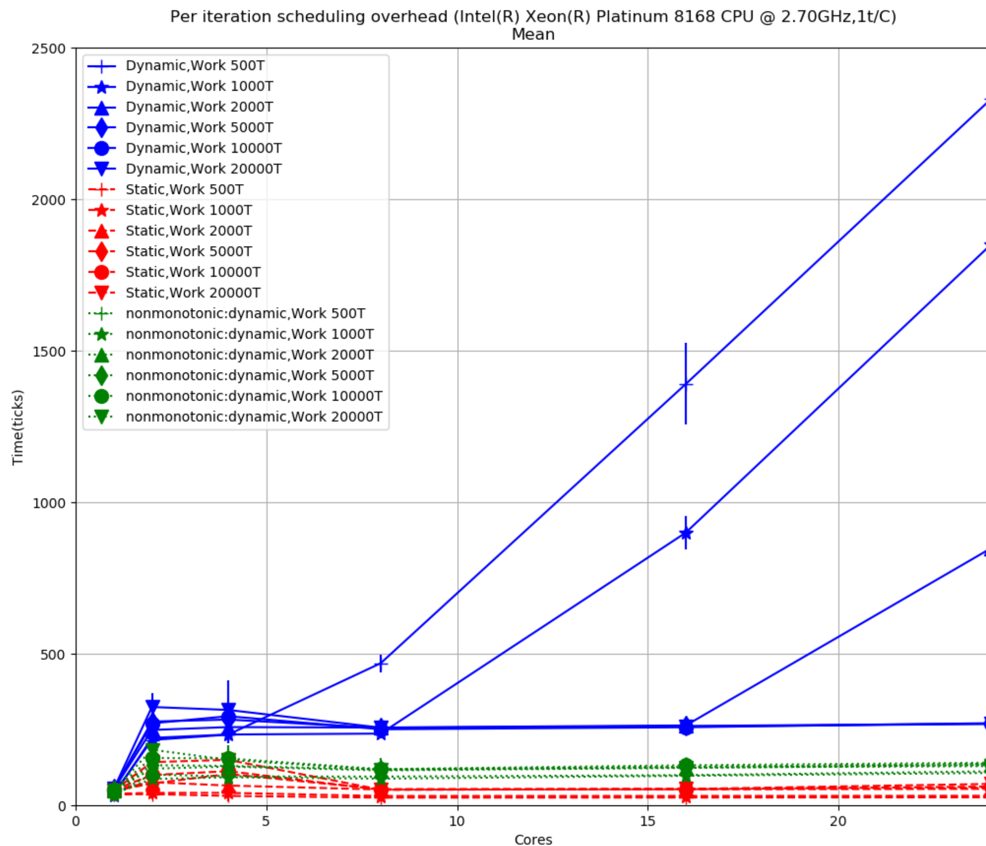
Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Why Should I Care?

A `nonmonotonic:dynamic` schedule can be implemented more efficiently than a monotonic one.

The graph shows time spent performing scheduling as we vary the work in each loop iteration and the number of threads.

With small amounts of work the cost of `monotonic:dynamic` scheduling can be large



Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Nonmonotonic Properties

Advantages

- Scheduling cost is significantly lower than `monotonic:dynamic`

  => Can be used on loops with little work in each iteration

- Gives an iteration distribution that is more like a blocked static one than a cyclic, dynamic one

  => Can improve cache efficiency

- Handles load imbalance better than a static schedule

Disadvantage

- Still more expensive that a static schedule

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

A real code example: MPAS-Ocean

“MPAS-Ocean is designed for the simulation of the ocean system from time scales of months to millenia and spatial scales from sub 1 km to global circulations.”

~230,000 lines of Fortran (77 and 90) code

MPI + OpenMP code, but run here on a single node (so no network communication occurs)

Our tests all use the “EC 60 to 30 km workload” running one simulated day with no I/O using MPAS v6.0

We are interested in the effect of different OpenMP loop schedules, not MPI vs OpenMP trade-offs

- Easy to change the OpenMP schedule since MPAS uses `schedule(runtime)` throughout

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

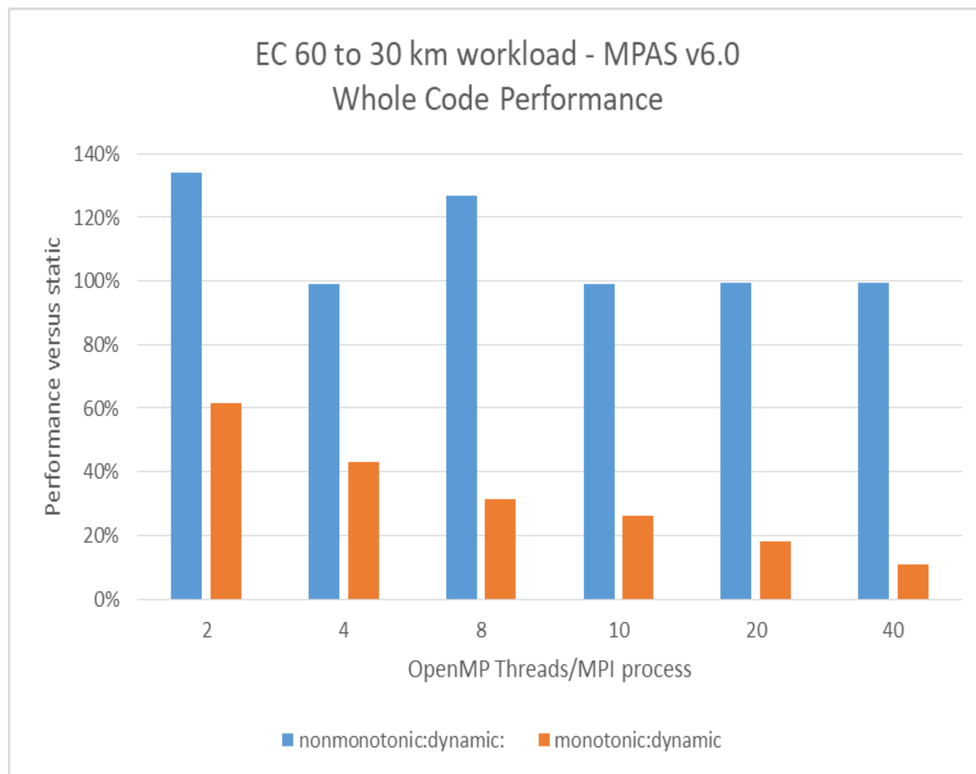
MPAS-O relative performance

No affinity specified

`nonmonotonic:dynamic` performs usefully better than static in a few cases and is marginally (1%) worse in others

Caveats

- Results may differ with other affinities (tuning affinity is a separate, if related, issue)
- This was enough to make 2T/MPI process better than one, but not more than that ☹



Full configuration details in backup section

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Nonmonotonic Schedule Conclusions

You need to understand this since it will be the default dynamic schedule in OpenMP 5.0

- Though implementations may play it safe and not enable it, so if you want it, say so in your code and tell compiler vendors to make it the default.

It can provide useful performance improvements with a small source code change

When you have load imbalance in a `!$omp do` reach for `schedule(nonmonotonic:dynamic)`, not just `schedule(dynamic)`

Compilers may still be catching up ☹

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Overall Conclusions

OpenMP continues to evolve and you need to pay that some attention

Small enhancements to the standard can be useful and easy to apply

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

BACKUP

Acknowledgements

MPAS-Ocean: Ringler, T., Petersen, M., Higdon, R. L., Jacobsen, D., Jones, P. W., & Maltrud, M. (2013). Ocean Modelling. Ocean Modelling, 69(C), 211–232. doi:10.1016/j.ocemod.2013.04.010

MPAS-O results created by Nitya Hariharan and Doug Jacobsen (both of Intel)

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Test Configuration Information

Performance results are based on testing as of September 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.

Lock Hint and Nonmonotonic Schedule Test Configuration:
Lock Hint testing by Intel as of 10 September 2018.
Nonmonotonic schedule testing by Intel as of 10 September 2018.
2x Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz, 192GB memory
OS: Red Hat EL 7.4, Compiler: icc (ICC) 19.0.0.070 20180524

MPAS-O Test Configuration:
Testing by Intel as of 31 August 2018.
2x Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz, 192 GB, 12x16GB 2666 MHz DDR4.
OS: Oracle Linux Server release 7.5, kernel version - 3.10.0-862.11.6.el7.crt1.x86_64
Compiler: icc (ICC) 18.0.3 20180410
MPAS-O:– v6.0 (`make CORE=ocean USE_OPENMP=true USE_PIO2=true`),PIO – v2.3.1,NetCDF – 4.4.1.1,PNetCDF – 1.9.0,HDF5 – 1.10.1
All runs are EC60to30km workload, 1 day simulation with I/O switched off.

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Legal Disclaimer & Optimization Notice

Performance results are based on testing as of September 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.

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. For more complete information visit www.intel.com/benchmarks.

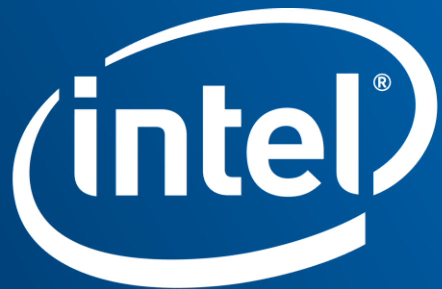
INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING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

Copyright © 2018, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Software