



Obtaining the Last Values of Conditionally Assigned Privates

**Hideki Saito, Serge Preis*,
Aleksei Cherkasov, Xinmin Tian**

Intel Corporation

(* at submission time)

2016/10/04 OpenMPCon2016



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.

Copyright © 2015, 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

Last Value through Lastprivate

```
#pragma omp simd lastprivate(x)
for(int i=0; i<N; i++){
    x = A[i];
    ...
}
```

// use of x.

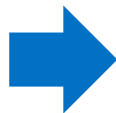
→ x is A[N-1].

```
#pragma omp simd lastprivate(x)
for(int i=0; i<N; i++){
    if (A[i]>0) {
        x = A[i];
        ...
    }
}
```

// use of x.

→ x is A[N-1] if A[N-1]>0,
else **unspecified**.

Unspecified isn't very useful.
Can we do better?



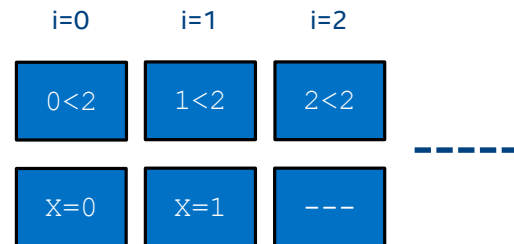
Problem

- The value from `lastprivate` (*list*) clause is unspecified if the list item is not assigned in the last iteration.
- Typically, what programmer wants under such circumstances is the value from the last iteration that actually performed the assignment.

```
#pragma omp simd lastprivate(x)
for(int i=0; i<N; i++){
    if (A[i]>0){ // may be FALSE for i=N-1.
        x = A[i];
        // possible use of x
    }
}
// use of x.
```

Does ORDERED construct help?

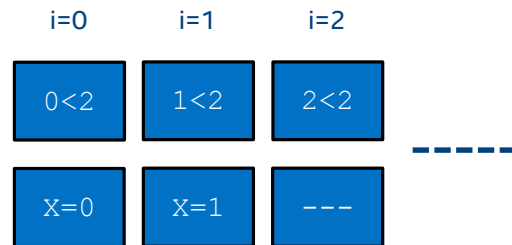
```
#pragma omp simd
for (i=0; i<N; i++) {
    if (i<2) {
#pragma omp ordered simd
        { x = i }
    } // x is 1 here. Good.
}
// use of x after the loop
```



X needs to be shared,
but it could work
for this case.....

Does ORDERED construct help? (cont)

```
#pragma omp simd
for (i=0; i<N; i++) {
    if (i<2) {
#pragma omp ordered simd
        { x = i }
    } // x is 1 here. Good.
    if (i<1) {
#pragma omp ordered simd
        { x = i }
    } // x is 0 again. Bad.
}
// use of x after the loop
```



Unfortunately, this doesn't go too far.

Why ORDERED Failed?

- Each ORDERED is individually processed.
 - Even if an ORDERED assigns at iter-N1, the next ORDERED can override it at iter-N2, where $N2 < N1$.
- ➔ Something at loop-level is needed to support multiple assignments.

How does ICC auto-vec handle it?

- Local reduce, per vector element, of last valid value and last assigned index
 - Global reduce after the end of the vector loop
- ➔ It's a bit more elaborate than typical reduction, but it almost look like reduction.
- ➔ Same concept applicable to threading? Absolutely!

How does ICC auto-vec handle it?

```
int foo(int *A, int N){
    int i, x = 0;
    for (i=0;i<N;i++){
        if (A[i]>0){
            x = A[i];
        }
    }
    return x;
}
```

```
..B1.13:
    vmovdqu  (%rdi,%rcx,4), %ymm4
    vpcmpgtd %ymm2, %ymm4, %ymm5
    vmovdqa  %ymm2, %ymm6
    vptest  %ymm1, %ymm5
    je      ..B1.15
..B1.14:
    vmovdqa  %ymm4, %ymm3
    vmovdqa  %ymm5, %ymm6
..B1.15:
    vmovmskps %ymm6, %r9d
    testl   %r9d, %r9d
    je      ..B1.17
```

Not exactly tracking the loop index value,
but vector code can afford to use
element position within vector register.

```
cmpq   %rsi, %rcx
jb     ..B1.13
```

Can we then call it a reduction???

```
#pragma omp simd reduction(=:x)
for(i=0;i<N;i++){
    if (cond(i)){
        x = ...
    }
}
// use of x after the loop
```

- Informal poll is showing that some people are allergic to calling it a reduction.
- Lastprivate() is already taken, and it's more efficient if it can be used.
 - We can't change lastprivate().
- Something that can coexist with lastprivate() is needed.

How about `lastvalue ()` clause?

```
#pragma omp simd lastvalue (x)
for(i=0;i<N;i++){
    if (cond(i)){
        x = ...
    }
}
// use of x after the loop
```

- Other than the name and lack of reduction op, it's the same as “assignment reduction.”
- Applicable to any construct where `lastprivate()` makes sense.
- Actual operation is reduction-like. As such, construct should also support reduction.

Should we support list item appearing as an R-value?

- If we were to call it a reduction, we wouldn't like to see it outside of reduction operation (which is assignment in this case).
- Some optimizations are easier to perform if we know R-value usage does not exist.
- Programmer can introduce new privates and mechanically get rid of R-value usages of the list items.
- Is this good enough reasons for not supporting R-value usage?
 - At least, this is worth explicitly discussing.

Can we have aggregates as list item?

- Yes, we should.
- However, unlike `lastprivate()`, `lastvalue()` has book-keeping overhead.
 - There is a limit to what we can feasibly do. Need to strike a right balance.
- Supporting something like below would be prohibitively expensive.

```
#pragma omp simd lastvalue(A)
for(i=0;i<N;i++){
    if (B[i]>0){ A[i] = ...}
    j = ...
    if (C[i]>0){ A[j] = ...}
}
```

What should we do with aggregates?

- List item as the unit of book-keeping
- No runtime check for matching against list item
- Whole array/struct as a list item should be supported
- Avoid supporting array subsection
- Support one array element as long as the subscript is compile time constant.
- Individual struct field should be supported

What's Next?

- Now that we defined the clause that keeps track of two values to perform something similar to reduction, can we do max location? Can we go more than 1-D?

```
for(i=0;i<N;i++){  
    if (max<A[i]){  
        max = A[i];  
        loc = i;  
    }  
}  
  
// use of loc.
```

Other Ideas We are Working On

- Vectorization of Indirect Calls
- Compress/Expand
- Conflict
- Loop w/ early exits
- Generalized Induction

Shameless Advertisement

- Today at 4:45pm: Panel Discussion
- Friday at 9:50am: Compress/Expand and Conflict

Summary of Proposal

1. `lastvalue` (*list*) clause syntax and spelling
2. Applicability same as `lastprivate` clause (`for`, `simd`, `for simd`, ...)
3. Aggregate variable as list item allowed, but subject to limitations to minimize book-keeping overhead.

