

OpenMP 4.0: a Significant
Paradigm Shift in High-level
Parallel Computing (Not just
Shared Memory anymore)

Michael Wong
OpenMP CEO



OpenMP 4.0: a Significant Paradigm Shift in High-level Parallel Computing (**Not just Shared Memory anymore**)

Michael Wong
michaelw@ca.ibm.com

OpenMP CEO
Chair of WG21 SG5 Transactional Memory
ISO C++ Standard, Head of Delegation for Canada and IBM

Acknowledgement and Disclaimer

- Numerous people internal and external, in industry and academia, have made contributions, influenced ideas, written part of this presentations, and offered feedbacks to form part of this talk.
- I even lifted this acknowledgement and disclaimer from some of them.
- But I claim all credit for errors, and stupid mistakes. **These are mine, all mine!**
- Any opinions expressed in this presentation are my opinions and do not necessarily reflect the opinions of IBM.

Legal Disclaimer

- ▣ This work represents the view of the author and does not necessarily represent the view of IBM.
- ▣ IBM, PowerPC and the IBM logo are trademarks or registered trademarks of IBM or its subsidiaries in the United States and other countries.
- ▣ Other company, product, and service names may be trademarks or service marks of others..

IBM Rational Disclaimer

- © Copyright IBM Corporation 2014. All rights reserved. The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. IBM, the IBM logo, Rational, the Rational logo, Telelogic, the Telelogic logo, and other IBM products and services are trademarks of the International Business Machines Corporation, in the United States, other countries or both. Other company, product, or service names may be trademarks or service marks of others.

OpenMP SIMD Subcommittee

- Technical leads
 - Xinmin Tian - Intel
- Active subcommittee members
 - Michael Klemm – Intel (courtesy for slides)
 - Alex Duran - BSC
 - Xavier Martotell- BSC
 - Kelvin Li - IBM
 - Nowal Copty – Oracle/Sun

What levels of parallelism is supported by OpneMP in today's hardware?

Cluster	Group of computers communicating through fast interconnect
Coprocessors/Accelerators	Special compute devices attached to the local node through special interconnect
Node	Group of processors communicating through shared memory
Socket	Group of cores communicating through shared cache
Core	Group of functional units communicating through registers
Hyper-Threads	Group of thread contexts sharing functional units
Superscalar	Group of instructions sharing functional units
Pipeline	Sequence of instructions sharing functional units
Vector	Single instruction using multiple functional units

SIMD Language Extensions

- IBM currently has 7 SIMD architectures
 - VMX, VMX128, VSX, SPE, BGL, BGQ, QPX
 - Each has its own proprietary language extension
 - Code written for one language extension can't be moved without a rewrite
 - We don't even have compatibility within our own company
- Outside our company Intel has 7 SIMD architectures
 - MMX, SSE, SSE2, SSE3, SSE4, AVX, AVX-512/MIC
 - MMX, SSEx, AVX each have a different language extension
 - Code written for one language extension can't be moved without a rewrite

“The Great Hope” - Auto-Vectorizing compilers

- SIMD floating point entered the market 10 years ago
 - (Intel Pentium III, Motorola G4, AMD K6-2)
- Software industry held its breath waiting for a “magic” auto-vectorizing compiler (including Microsoft)
- Despite 15 years of research and development the industry still doesn’t have a good auto-vectorizing compiler
- Industry instead ended up with primitive language support
 - Multiple non-compatible language extensions
 - Compiler intrinsics
- Using intrinsics humans still produce superior vector code but at great pain

Why autovectorization fail?

- SIMD register width has increased from 128-256-512, 1024 soon
- Instructions are more powerful and complex
 - Hard for compiler to select proper instruction
 - Code pattern needs to be recognized by the compiler
 - Precision requirements often inhibit SIMD codegen

What sort of loops can be vectorized?

- Countable
- Single entry, single exit
- Straight-line code
- Innermost loop of a nest
- No function calls
- Certain non-contiguous memory access
- Some Data dependencies
- Efficient Alignment
- Mixed data types
- Non-unit stride between elements
- Loop body too complex (register pressure)

Industry needs better language support for SIMD

- 80% of Cell programmers time spent vectorizing code
- Need to reduce programming effort
 - Fewer code modifications to vectorize
 - Rapid conversion of scalar to vector code
- Code portability
 - Don't rewrite for every SIMD architecture
- Less code maintenance
 - Intrinsics impossible to maintain
 - Easier to rewrite than figuring out what the code is doing
- Support required vendor-specific extensions before OpenMP 4.0

SIMD Loop Construct

- Vectorize a loop nest

- Cut loop into chunks that fit a SIMD vector register
- No parallelization of the loop body

- Syntax (C/C++)

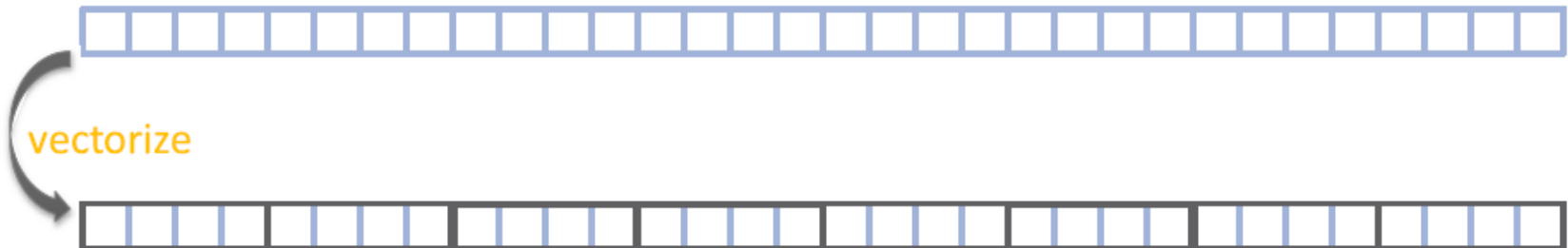
```
#pragma omp simd [clause[[, clause],...]  
for-loops
```

- Syntax (Fortran)

```
!$omp simd [clause[[, clause],...]  
do-loops
```

Sum product Example

```
void sprod(float *a, float *b, int n) {  
    float sum = 0.0f;  
    #pragma omp simd reduction(+:sum)  
    for (int k=0; k<n; k++)  
        sum += a[k] * b[k];  
    return sum;  
}
```



Data Sharing Clauses

- `private (var-list) :`

Uninitialized vectors for variables in *var-list*



- `firstprivate (var-list) :`

Initialized vectors for variables in *var-list*



- `reduction (op: var-list) :`

Create private variables for *var-list* and apply reduction operator *op* at the end of the construct



SIMD Loop Clause

■ `safelen (length)`

- Maximum number of iterations that can run concurrently without breaking a dependence
- in practice, maximum vector length

■ `linear (list[:linear-step])`

- The variable's value is in relationship with the iteration number
 - $x_i = x_{\text{orig}} + i * \text{linear-step}$

■ `aligned (list[:alignment])`

- Specifies that the list items have a given alignment
- Default is alignment for the architecture

■ `collapse (n)`

SIMD Worksharing Construct

- Parallelize and vectorize a loop nest

- Distribute a loop's iteration space across a thread team
- Subdivide loop chunks to fit a SIMD vector register

- Syntax (C/C++)

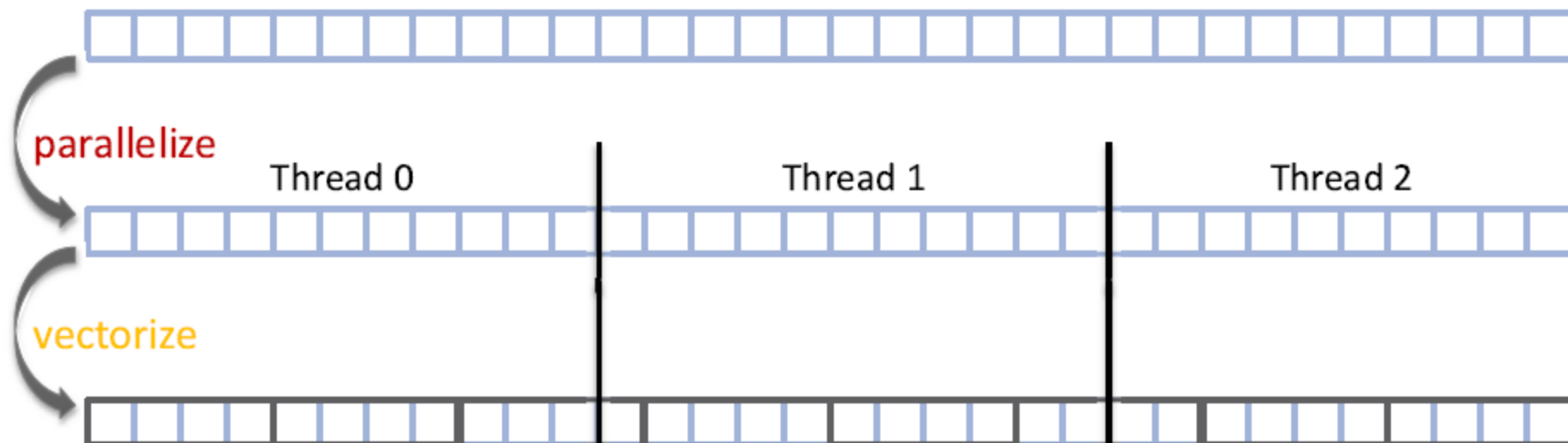
```
#pragma omp for simd [clause[[, clause],...]  
for-loops
```

- Syntax (Fortran)

```
!$omp do simd [clause[[, clause],...]  
do-loops
```

Parallelize and vectorize

```
void sprod(float *a, float *b, int n) {  
    float sum = 0.0f;  
    #pragma omp for simd reduction(+:sum)  
    for (int k=0; k<n; k++)  
        sum += a[k] * b[k];  
    return sum;  
}
```



SIMD Function Vectorization

```
float min(float a, float b) {  
    return a < b ? a : b;  
}  
  
float distsq(float x, float y) {  
    return (x - y) * (x - y);  
}  
  
void example() {  
    #pragma omp parallel for simd  
    for (i=0; i<N; i++) {  
        d[i] = min(distsq(a[i], b[i]), c[i]);  
    }  
}
```

SIMD Function Vectorization

- Declare one or more functions to be compiled for calls from a SIMD-parallel loop

- Syntax (C/C++):

```
#pragma omp declare simd [clause[[, clause],...]  
[#pragma omp declare simd [clause[[, clause],...]  
...  
function-definition-or-declaration
```

- Syntax (Fortran):

```
!$omp declare simd (proc-name-list)
```

SIMD Function Vectorization

```
#pragma omp declare simd
```

```
float min(float a, float b) {  
    return a < b ? a : b;  
}
```

```
vec8 min_v(vec8 a, vec8 b) {  
    return a < b ? a : b;  
}
```

```
#pragma omp declare simd
```

```
float distsq(float x, float y) {  
    return (x - y) * (x - y);  
}
```

```
vec8 distsq_v(vec8 x, vec8 y)  
    return (x - y) * (x - y);  
}
```

```
void example() {
```

```
#pragma omp parallel for simd
```

```
    for (i=0; i<N; i++) {  
        d[i] = min(distsq(a[i], b[i]), c[i]);  
    } }
```

```
vd = min_v(distsq_v(va, vb), vc)
```

SIMD Function Vectorization

- `simdlen (length)`

→ generate function to support a given vector length

- `uniform (argument-list)`

→ argument has a constant value between the iterations of a given loop

- `inbranch`

→ function always called from inside an if statement

- `notinbranch`

→ function never called from inside an if statement

- `linear (argument-list[:linear-step])`

- `aligned (argument-list[:alignment])`

- `reduction (operator:list)`




Same as before

inbranch

```
#pragma omp declare simd inbranch
```

```
float do_stuff(float x) {  
    /* do something */  
    return x * 2.0;  
}
```


```
vec8 do_stuff_v(vec8 x, mask m) {  
    /* do something */  
    vmulpd x{m}, 2.0, tmp  
    return tmp;  
}
```



```
void example() {  
    #pragma omp simd
```

```
    for (int i = 0; i < N; i++)  
        if (a[i] < 0.0)  
            b[i] = do_stuff(a[i]);  
}
```

```
for (int i = 0; i < N; i+=8) {  
    vcmp_lt &a[i], 0.0, mask  
    b[i] = do_stuff_v(&a[i], mask);  
}
```



My blogs and email address

- **ISOCPP.org Director, VP**
<http://isocpp.org/wiki/faq/wg21#michael-wong>
OpenMP CEO: <http://openmp.org/wp/about-openmp/>
My Blogs: <http://ibm.co/pCvPHR>
C++11 status: <http://tinyurl.com/43y8xgf>
Boost test results
<http://www.ibm.com/support/docview.wss?rs=2239&context=SSJT9L&uid=swg27006911>
C/C++ Compilers Feature Request Page
http://www.ibm.com/developerworks/rfe/?PROD_ID=700
Chair of WG21 SG5 Transactional MemoryM:
<https://groups.google.com/a/isocpp.org/forum/?hl=en&fromgroups#!forum/tm>

FRAGEN?

Ich freue mich auf Ihr Feedback!

Vielen Dank!

Michael Wong