

A proposal for User-Defined Reductions in OpenMP

A. Duran¹, R. Ferrer¹, M. Klemm², B. de Supinski³, E. Ayguadé¹

¹BSC, ²Intel, ³LLNL

June 16th 2010

Outline

- 1 Motivation
- 2 UDR Design rationale
- 3 Declaring UDRs
- 4 Array reductions
- 5 C++ specific extensions
- 6 Conclusions

Outline

- 1 Motivation
- 2 UDR Design rationale
- 3 Declaring UDRs
- 4 Array reductions
- 5 C++ specific extensions
- 6 Conclusions

Reductions in OpenMP 3.0

Current OpenMP supports reduction:

- basic scalar types
- simple arithmetic operators (+, -, *, &, ...)
- array reductions (Fortran only)
- min and max operators (Fortran only)

Reductions in OpenMP 3.0

Current OpenMP supports reduction:

- basic scalar types
- simple arithmetic operators (+, -, *, &, ...)
- array reductions (Fortran only)
- min and max operators (Fortran only)

Users with other reductions must find their way manually:

- Using **critical**
- Using **atomic** (when possible)
- Adding complex code to implement the *reduction*

Reductions by hand

```
1 complex_t complex_mul(complex_t a, complex_t b);
2
3 void example(complex_t *array, size_t N) {
4     complex_t prd = {1.0, 0.0};
5
6
7
8
9     #pragma omp parallel reduction(prd)
10    {
11
12        #pragma omp for
13        for (size_t i = 0; i < N; i++)
14            prd = complex_mul(prd, array[i]);
15
16    }
17
18
19 }
```

Reductions by hand

```
1 complex_t complex_mul(complex_t a, complex_t b);
2
3 void example(complex_t *array, size_t N) {
4     complex_t prd = {1.0, 0.0};
5
6     int nthreads = omp_get_max_threads();
7     complex_t part_prd[nthreads];
8
9     #pragma omp parallel shared(part_prd) private(prd)
10    {
11        prd = {1.0, 0.0};
12        #pragma omp for
13        for (size_t i = 0; i < N; i++)
14            prd = complex_mul(prd, array[i]);
15        part_prd[omp_get_thread_num()] = prd;
16    }
17
18    for (int thr = 0; thr < nthreads; thr++)
19        prd = complex_mul(prd, part_prd[thr]);
20
21 }
```

Not good

Drawbacks

- More complex user code
- Error prone
- Doesn't benefit from implementation improvements

Not good

Drawbacks

- More complex user code
- Error prone
- Doesn't benefit from implementation improvements

Solution

Add user-defined reductions to OpenMP

User-defined reductions

Allow the user to inform OpenMP about new reductions by providing:

- A type
- An operation over that type
- The identity value for that operation and type

User-defined reductions

Allow the user to inform OpenMP about new reductions by providing:

- A type
- An operation over that type
- The identity value for that operation and type

Presented to the OpenMP language committee (still on discussion)

Outline

- 1 Motivation
- 2 UDR Design rationale**
- 3 Declaring UDRs
- 4 Array reductions
- 5 C++ specific extensions
- 6 Conclusions

Driving design goals

- follow OpenMP directive-based philosophy
- support all OpenMP base languages
 - but maintain a common syntax as much as possible
- follow a declaration/usage pattern
- Allow code re-use
- Allow efficient implementation
 - require associativity and commutativity

Outline

- 1 Motivation
- 2 UDR Design rationale
- 3 Declaring UDRs**
- 4 Array reductions
- 5 C++ specific extensions
- 6 Conclusions

Declaring an UDR

Syntax

C:

```
1 #pragma omp declare reduction( operator-list : type ) [ clause ]
```

C++:

```
1 #pragma omp declare reduction([template <template-params>] operator-list : type-list) [clause]
```

Fortran:

```
1 !$omp declare reduction( operator-list : typename ) [clause]
```

where clause is:

```
1 identity( expression | brace-initializerC/C++ | constructor [( argument-list )]C++ )
```

Example

```
1 complex_t complex_mul(complex_t a, complex_t b);
2
3 #pragma omp declare reduction(complex_mul : complex_t) \
4     identity({1.0, 0.0})
5
6 void example(complex_t *array, size_t N) {
7     complex_t prd = {1.0, 0.0};
8
9     #pragma omp parallel for reduction(complex_mul:prd)
10        for (size_t i = 0; i < N; i++)
11        prd = complex_mul(prd, array[i]);
12
13 }
```

Operator requisites

- binary operators with compatible arguments with UDR type
 - Return value either by return or parameter (*,&)
 - Return value priority: function return, left parameter, right parameter
 - Allow const and references
- (C++) Unary member functions
 - specified by prepending a dot to the operator name
- (C++) valid overloaded operators
- associative
- commutative
- available using base language “symbol“ lookup
 - both at declaration and usage

Identity value

- By default:
 - C/Fortran Zero initialization
 - C++ C++ rules for value-initialization
- Can be overridden by the **identity** clause
 - The identity expression must evaluate always to the same value
 - An implementation can evaluate it one or more times

Outline

- 1 Motivation
- 2 UDR Design rationale
- 3 Declaring UDRs
- 4 Array reductions**
- 5 C++ specific extensions
- 6 Conclusions

Array UDRs

Declaration

Types can be prepended with `[]` that indicate that is going to be an array UDR

- One `[]` per dimension
- Dimension size is not fix at declaration
 - Operators can have additional integer parameters to get the actual size

Usage

Array UDRs can be applied to variables of:

- Array types
- Pointer to array types
 - allows to "recover" arrays through function calls

Support for VLA (pointers to) arrays is limited to C.

Example

```

1  const int N = 10;
2  void vector_add ( int *A, int *B, int n );
3
4  #pragma omp declare reduction ( vector_add : int[] )
5
6  void foo ( int * a, int * b, int n )
7  {
8      int v1[N];
9      int (*v2) [N] = (int (*) [N]) a;
10     int (*v3) [n] = (int (*) [n]) b; // VLA; Only valid in C
11
12     #pragma omp for reduction ( vector_add : v1, v2, v3 )
13         for ( ... ) { ... }
14 }

```

Outline

- 1 Motivation
- 2 UDR Design rationale
- 3 Declaring UDRs
- 4 Array reductions
- 5 C++ specific extensions**
- 6 Conclusions

Constructed identities

The special **constructor** keyword can be used in the **identity** clause

- Private copies will be initialized with a constructor instead of by-assignment

Constructed identities

The special **constructor** keyword can be used in the **identity** clause

- Private copies will be initialized with a constructor instead of by-assignment

```
1 #pragma omp declare reduction( * : Complex) identity(constructor(1.0,0.0))
```

Inheritance support

Idea

Support the C++ philosophy of allowing to use methods defined over base classes with derived classes

Inheritance support

Idea

Support the C++ philosophy of allowing to use methods defined over base classes with derived classes

- UDR of base classes can be used for derived classes
- Only if it does not create object slicing
 - operator parameters must be pointers or references

Template UDRs

Allows to declare an UDR on all (or partial) instantiations of template type

Template UDRs

Allows to declare an UDR on all (or partial) instantiations of template type

```

1 #pragma omp declare \
2     reduction(template<class T> std::list<T>::merge : std::list<T> )
3
4 void foo ( )
5 {
6     std::list<int> li;
7     std::list<float> lf;
8
9     #pragma omp parallel for reduction(std::list<int>::merge : li )
10        reduction(std::list<float>::merge : lf )
11        for ( ... ) { ... }
12 }

```

Dot syntax

Simplify qualification for C++ by doing auto-qualification based on the UDR type

Dot syntax

Simplify qualification for C++ by doing auto-qualification based on the UDR type

```

1 /* #pragma omp declare \
2     reduction(template<class T> .merge : std::list<T> ) */
3
4 #pragma omp declare \
5     reduction( .merge : std::list<int>,std::list<float> )
6
7 void foo ( )
8 {
9     std::list<int> li;
10    std::list<float> lf;
11
12    #pragma omp parallel for reduction( .merge : li , lf )
13        for ( ... ) { ... }
14 }

```

Outline

- 1 Motivation
- 2 UDR Design rationale
- 3 Declaring UDRs
- 4 Array reductions
- 5 C++ specific extensions
- 6 Conclusions**

Conclusions

- Proposal allows users to *extend* OpenMP with their own reduction operations
 - aggregated types
 - arrays
 - template types
- Still in discussion for OpenMP 3.1
 - We're looking for user codes

The End

Thanks for your attention!

Min/Max

```
1 #pragma omp declare \  
2     reduction (template<typename T> std::min : T) \  
3     identity(std::numeric_limits<T>::max())  
4 #pragma omp declare \  
5     reduction (template<typename T> std::max : T) \  
6     identity(std::numeric_limits<_T>::min())
```

Some example

```

1 inline void pointMin ( Point *a, Point *b )
2 {
3     if ( a->x < b->x ) a->x = b->x;
4     if ( a->y < b->y ) a->y = b->y;
5 }
6
7 inline void pointMax ( Point *a, Point *b )
8 {
9     if ( a->x > b->x ) a->x = b->x;
10    if ( a->y > b->y ) a->y = b->y;
11 }
12
13 #pragma omp declare reduction (pointMax: Point *)
14 #pragma omp declare reduction (pointMin: Point *)
15
16 void findMinMax(PointVector points, index_t n, Point* minPoint,
17               Point* maxPoint)
18 {
19
20 #pragma omp parallel for schedule(static) \
21     reduction(pointMin:minPoint) reduction(pointMax:maxPoint)
22     for (index_t i = 1; i < n; i++) {
23         if (points[i].x < min.x) minPoint->x = points[i].x;
24         if (points[i].y < min.y) minPoint->y = points[i].y;
25         if (points[i].x > max.x) maxPoint->x = points[i].x;
26         if (points[i].y > max.y) maxPoint->y = points[i].y;
27     }
28 }
29 }

```

Gromacs

```

1 void array_rvec_add ( rvec *a, rvec *b, int n )
2 {
3     for ( int i = 0; i < n ; i++ ) {
4         a[i].xx += b[i].xx;
5         a[i].yy += b[i].yy;
6         a[i].zz += b[i].zz;
7     }
8 }
9
10 void real_array_add ( real *a, real *b, int n)
11 {
12     for ( int i = 0; i < n ; i++ ) a[i] += b[i];
13 }
14
15 #pragma omp declare reduction (array_rvec_add:rvec[]) identity({0.0,0.0,0.0})
16 #pragma omp declare reduction (array_real_add:real[]) identity(0.0)
17
18 ...
19 neg2 = mdatoms->nenergrp*mdatoms->nenergrp;
20
21 rvec (*pf) [mdatoms->nr] = (rvec (*) [mdatoms->nr]) f;
22 rvec (*pfshift) [SHIFTS] = (rvec (*) [SHIFTS]) pfshift;
23 real (*pegcoul) [neg2] = (real (*) [neg2]) egcoul;
24 real (*pegnb) [neg2] = (real (*) [neg2]) egnb;
25
26 #pragma omp parallel for reduction(array_rvec_add:pf, pfshift) \
27                               reduction(array_real_add:pegcoul, pgnb)
28     for(b=0; b<nb; b++)
29         function(pfshift, pf, pegcoul, pgnb);
30
31 ...

```