The OpenMP Memory Model

Jay Hoeflinger

Bronis de Supinski

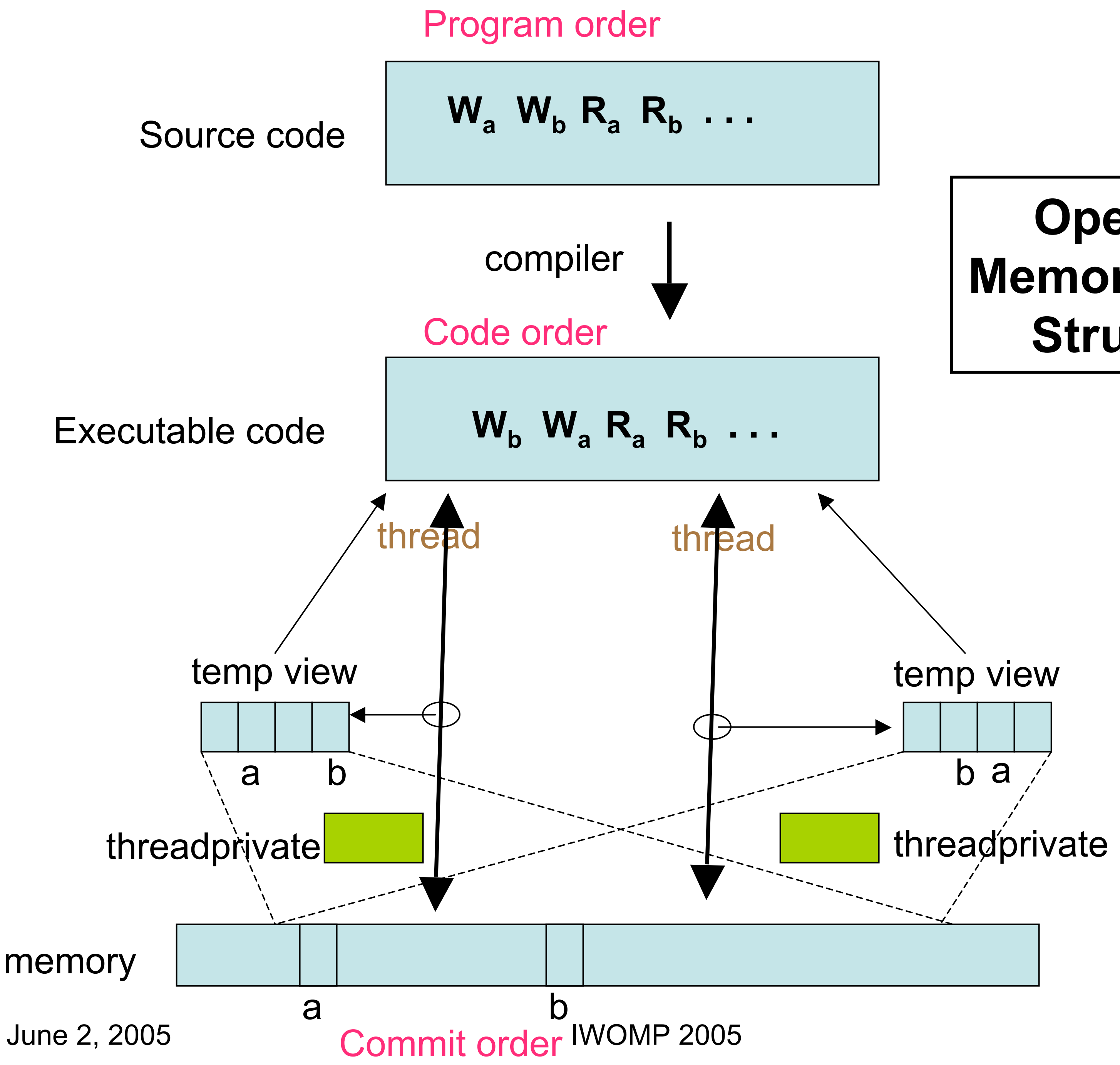
Memory Model in Prior Specs

- No separate section
- Scattered in Execution Model, Flush description, data sharing attribute section
- Unclear, implied

OpenMP Memory Model in 2.5

- Model Structure
  - Parts of the model
  - Shared & private access
  - Memory coherence
  - X-thread access: private
- Flush in OpenMP
  - Relaxed consistency
  - Flush operation
  - Flush guarantees consist.
  - Volatile relates to flush
- Memory consistency
  - Formal memory consist.
  - Memory consist. of flush
  - Flush operation specified with flush directive

OpenMP Memory Model Structure



Program order

Source code

W_a W_b R_a R_b . . .

compiler

Code order

Executable code

W_b W_a R_a R_b . . .

thread

thread

temp view

a b

temp view

b a

threadprivate

threadprivate

memory

a

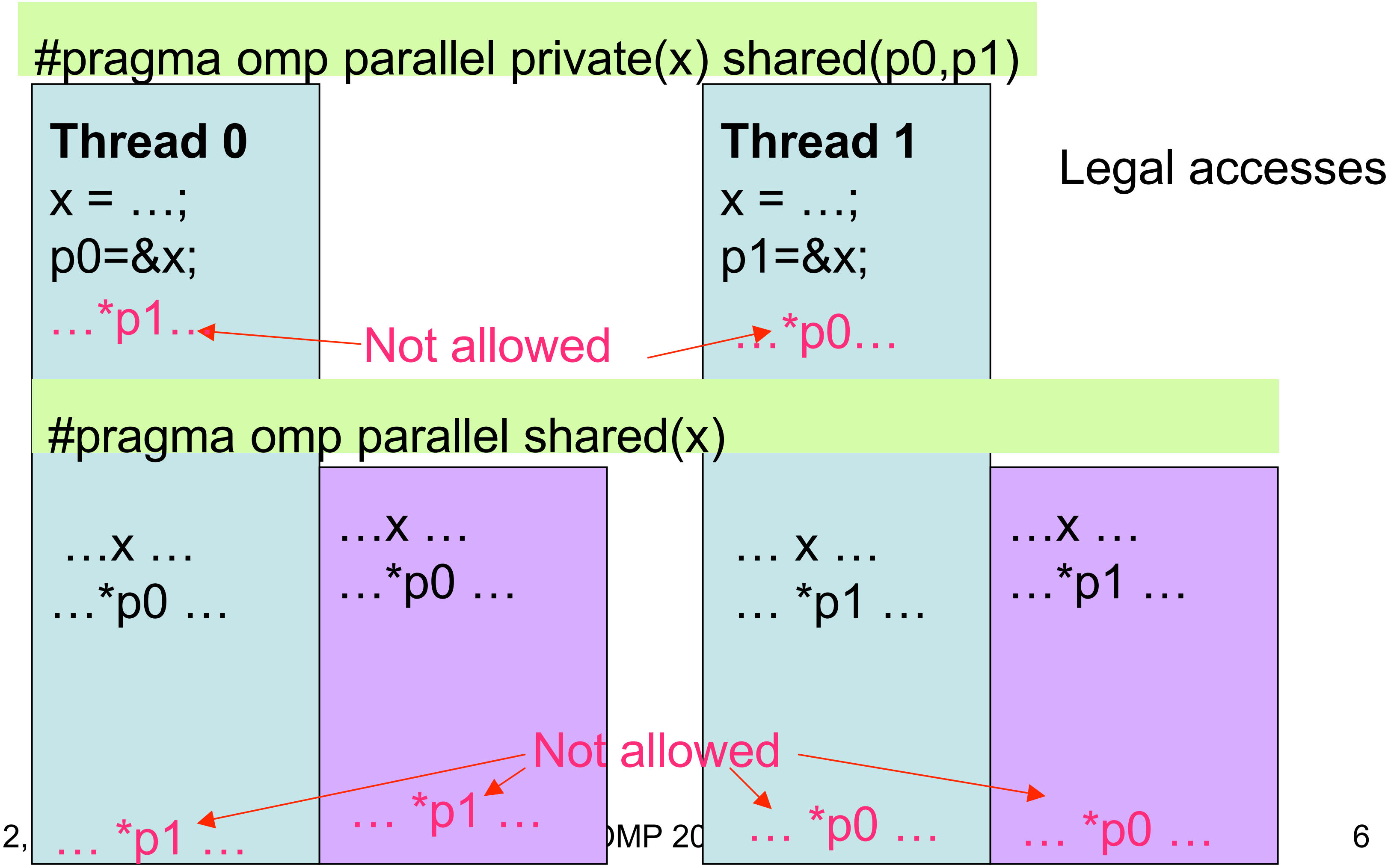
b

Commit order

Shared and Private Access

- All shared and private variables have original variables
- Shared access to a variable:
  - Within the structured block, references to the variable all refer to the original variable
- Private access to a variable:
  - A variable of the same type and size as the original variable is provided for each thread

Rules about cross-thread private access



#pragma omp parallel private(x) shared(p0,p1)

Thread 0

```
x = …;
p0=&x;
…*p1…
```

Thread 1

```
x = …;
p1=&x;
…*p0…
```

Legal accesses

Not allowed

#pragma omp parallel shared(x)

```
…x …
…*p0 …
… *p1 …
```

```
…x …
…*p0 …
… *p1 …
```

```
… x …
… *p1 …
… *p0 …
```

```
…x …
…*p1 …
… *p0 …
```

Not allowed

Flush Is the Key OpenMP Operation

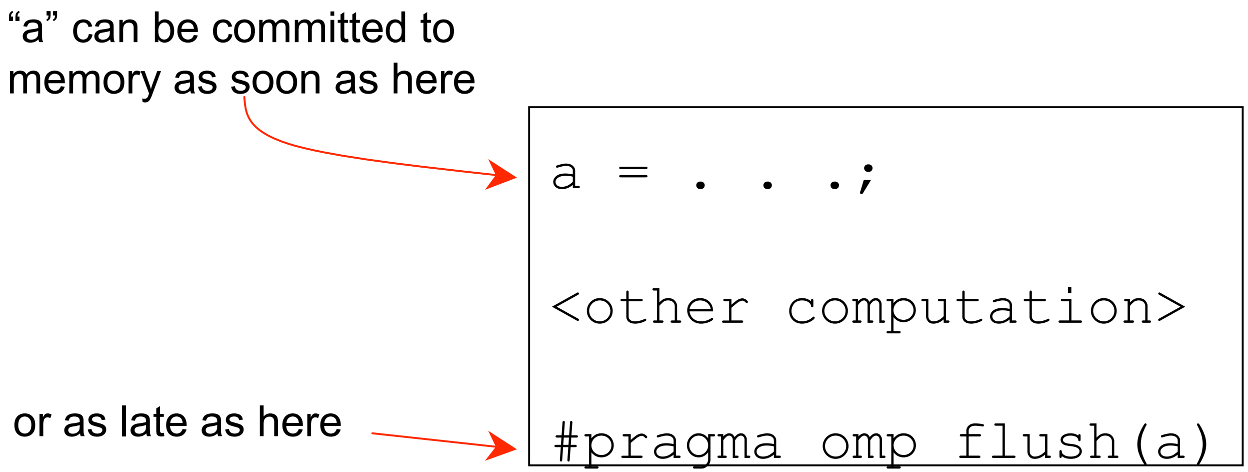
`Flush operation:  flush` ***`flush-set`***

- Prevents re-ordering of accesses
- Provides a guarantee that memory references are complete
- Provides the mechanism for moving data between threads
- Allows for overlapping computation with communication

Implicit flushes

- In barriers
- At entry to and exit from
  - Parallel, parallel worksharing, critical, ordered regions
- At exit from worksharing regions (unless `nowait` is specified)
- In `omp_set_lock`, `omp_set_nest_lock`, `omp_set_nest_lock`, `omp_unset_nest_lock`
- In `omp_test_lock`, `omp_test_nest_lock`, if lock is acquired
- At entry to and exit from `atomic -` flush-set is the address of the variable atomically updated

Temporary View Allows Hiding Memory Latency



“a” can be committed to memory as soon as here

or as late as here

```
a = . . .;

<other computation>

#pragma omp flush(a)
```

Re-ordering Example

```
a = ...;  //(1)
b = ...;  //(2)
c = ...;  //(3)

#pragma omp flush(c)   //(4)
#pragma omp flush(a,b) //(5)

. . . a . . . b . . .; //(6)
. . . c . . .;         //(7)
```

(1) and (2) may not be moved after (5).

(6) may not be moved before (5).

(4) and (5) may be interchanged at will.

Moving data between threads

- To move the value of a shared var from thread a to thread b, do the following in exactly this order:
  - Write var on thread a
  - Flush var on thread a
  - Flush var on thread b
  - Read var on thread b

But Explicit Flush is HARD to Use Correctly

Acknowledgement: Yuan Lin, Sun Microsystems

```
Producer:

data = produce_new
!$omp flush(data)
flag = 1
!$omp flush(flag)


Consumer:

flag = 0
do
  !$omp flush(flag)
while (flag .eq. 0)
!$omp flush(data)
consume_data = data
```

```
Producer:

data = produce_new
!$omp flush(data, flag)
flag = 1
!$omp flush(flag)


Consumer:

flag = 0
do
  !$omp flush(flag)
while (flag .eg. 0)
!$omp flush(flag, data)
consume_data = data
```

Sequential Consistency

- In a multi-processor, ops are sequentially consistent if
  - Commit order == program order in each thread
  - Same overall order seen on all threads

program order == code order == commit order

Weak Ordering

- Memory ops must be divided into “data” ops and “synch” ops
- Data ops (reads & writes) are not ordered w.r.t. each other
- Data ops **are** ordered w.r.t. synch ops and synch ops are ordered w.r.t. each other

OpenMP ordering ~= weak ordering

- OpenMP re-ordering restrictions amount to weak ordering with “flush” identified as a “synch” op.
- But, it’s weaker than weak ordering.

Relaxed memory model enables use of NUMA machines – especially cluster implementations of OpenMP

OpenMP Locks and Flush

- Is a flush implied for OpenMP lock routines?
- Fortran 2.0 is silent, but lock routines are not included on list of places where flush is implied
- C/C++ 2.0 also silent, but
  - "There may be a need for flush directives to make the values of other variables consistent."
- Various people on previous committees say the answer is "no".
- But, people have not gotten the message

Typical OpenMP lock code

```
!$omp parallel

        . . .

      call omp_set_lock(lock)
      count = count + 1
      call omp_unset_lock(lock)

      . . .

!$ omp end parallel
```

```
!$omp parallel

        . . .

      call omp_set_lock(lock)
!$omp flush(count)
      count = count + 1
!$omp flush(count)
      call omp_unset_lock(lock)

      . . .

!$ omp end parallel
```

Required if lock routines do not imply flush

Example of incorrect code: SPEC OpenMP Code ammp

```
#ifdef _OPENMP
omp_set_lock(&(a1->lock));
#endif
a1fx = a1->fx;
a1fy = a1->fy;
a1fz = a1->fz;
a1->fx = 0;
a1->fy = 0;
a1->fz = 0;
xt = a1->dx*lambda +a1->x - a1->px;
yt = a1->dy*lambda +a1->y - a1->py;
zt = a1->dz*lambda +a1->z - a1->pz;
#ifdef _OPENMP
omp_unset_lock(&(a1->lock));
#endif
```

Summary

- In 2.5, memory model is explicit
- Cross-thread private access rules
- Description of flush and how to use
- Relates OpenMP consistency to formal consistency models
- Locks imply no-list flush