Loop Parallelism with OpenMP

Riddle

Which assertion is right?

1. Processes have private heaps, and threads share a stack.
2. Processes share a heap, and threads have private stacks.
3. Processes have private heaps, and threads have private stacks.
4. Processes share a heap, and theads share a stack.
5. Processes are for lawyers, and you have no idea of this stack-heap thing.

From Program To Process (1)

```
int x;
int i, n;
n = 10;
x = 1;
for (i=0 ; i<n ; i++) {
  x = x * 2;
}
```

Source code

Assembly code

```
    movl   $10, -12(%rbp)
    movl   $1, -4(%rbp)
    movl   $0, -8(%rbp)
    jmp    L2
L3: sall   -4(%rbp)
    incl   -8(%rbp)
L2: movl   -8(%rbp), %eax
    cmpl   -12(%rbp), %eax
    jl     L3
    movl   -4(%rbp), %edx
    ……
```

From Program To Process (2)

```
    movl   $10, -12(%rbp)
    movl   $1, -4(%rbp)
    movl   $0, -8(%rbp)
    jmp    L2
L3: sall   -4(%rbp)
    incl   -8(%rbp)
L2: movl   -8(%rbp), %eax
    cmpl   -12(%rbp), %eax
    jl     L3
    movl   -4(%rbp), %edx
    ……
```

```
    movl   $10, -12(%rbp)
    movl   $1, -4(%rbp)
    movl   $0, -8(%rbp)
    jmp    L2
L3: sall   -4(%rbp)
    incl   -8(%rbp)
L2: movl   -8(%rbp), %eax
    cmpl   -12(%rbp), %eax
    jl     L3
    movl   -4(%rbp), %edx
    ……
```

From Program To Process (3)

OPCODE SRC DEST
4 + 8 + 8
movl = 0010
Jmp = 0100
jl = 0110
sall = 1000
incl = 0011
cmpl = 0101

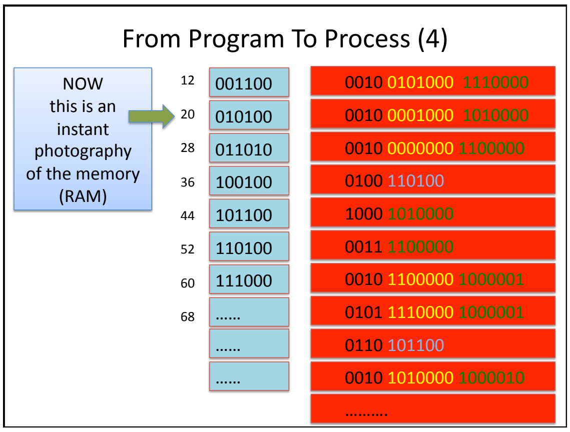
direct value = 0 + value + 00
1 + 4 + 2
indirect = 1 + desloc + reg.
1 + 4 + 2
rbp = 00
eax = 01
edx = 10

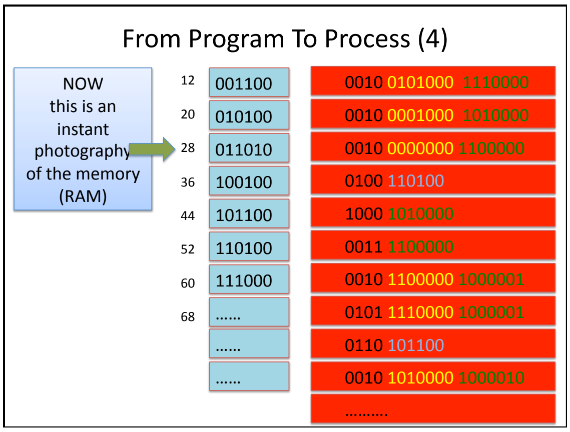
0000	0010 0101000 1110000
0001	0010 0001000 1010000
0002	0010 0000000 1100000
0003	0100 0006
0004	1000 1010000
0005	0011 1100000
0006	0010 1100000 1000001
0007	0101 1110000 1000001
0008	0110 0004
0009	0010 1010000 1000010
0010	……

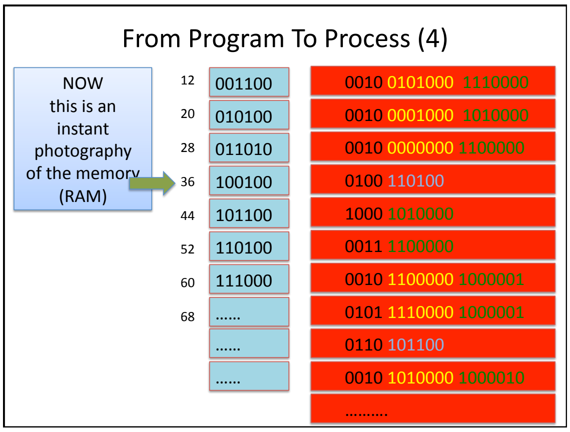
From Program To Process (4)

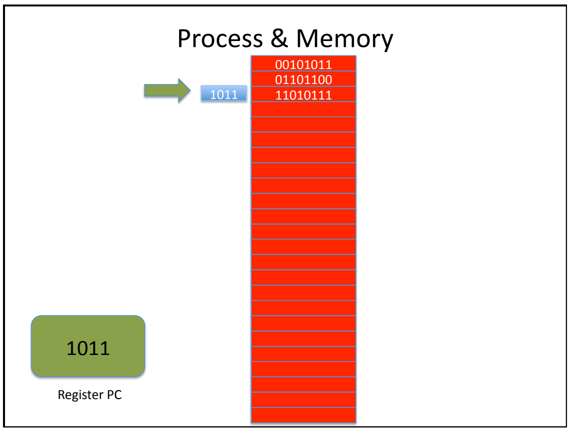
NOW this is an instant photography of the memory (RAM)

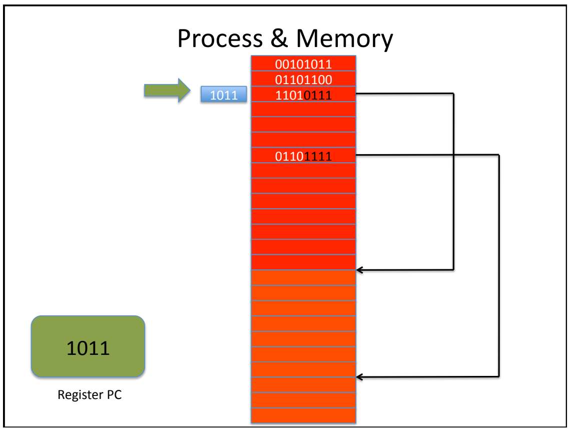
12	001100	0010 0101000 1110000
20	010100	0010 0001000 1010000
28	011010	0010 0000000 1100000
36	100100	0100 110100
44	101100	1000 1010000
52	110100	0011 1100000
60	111000	0010 1100000 1000001
68	……	0101 1110000 1000001
	……	0110 101100
	……	0010 1010000 1000010
		……

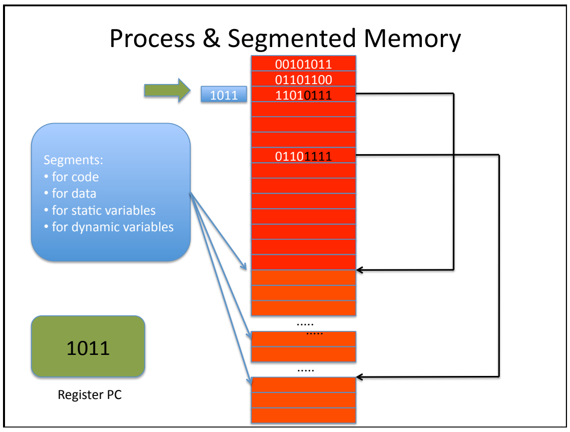

From Program To Process (4)
NOW this is an instant photography of the memory (RAM)
12 001100
20 010100
28 011010
36 100100
44 101100
52 110100
60 111000
68 ......
......
......
0010 0101000 1110000
0010 0001000 1010000
0010 0000000 1100000
0100 110100
1000 1010000
0011 1100000
0010 1100000 1000001
0101 1110000 1000001
0110 101100
0010 1010000 1000010
....


From Program To Process (4)
NOW this is an instant photography of the memory (RAM)
12 001100
20 010100
28 011010
36 100100
44 101100
52 110100
60 111000
68 ......
......
......
0010 0101000 1110000
0010 0001000 1010000
0010 0000000 1100000
0100 110100
1000 1010000
0011 1100000
0010 1100000 1000001
0101 1110000 1000001
0110 101100
0010 1010000 1000010
....

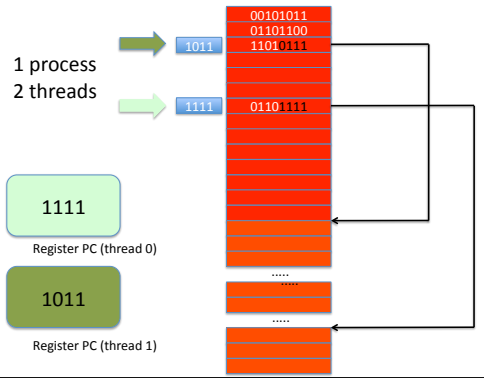

From Program To Process (4)
NOW this is an instant photography of the memory (RAM)
12 001100
20 010100
28 011010
36 100100
44 101100
52 110100
60 111000
68 ......
......
......
0010 0101000 1110000
0010 0001000 1010000
0010 0000000 1100000
0100 110100
1000 1010000
0011 1100000
0010 1100000 1000001
0101 1110000 1000001
0110 101100
0010 1010000 1000010
....


Process & Memory
1011
00101011
01101100
11010111
1011
Register PC


Process & Memory
1011
00101011
01101100
11010111
01101111
1011
Register PC


Process & Segmented Memory
1011
00101011
01101100
11010111
01101111
Segments:
• for code
• for data
• for static variables
• for dynamic variables
.....
.....
1011
Register PC

Process & Threads



Threads and segments

- Threads inside a process share the code segment.
  - A private PC points to a given instruction for each thread.
- Threads share the heap segment.
  - All dynamic vars are shared.
- Threads have private stack segments.
  - Vars local to a procedure are private.

Riddle

Which assertion is right?

1. Processes have private heaps, and threads share a stack.
2. Processes share a heap, and threads have private stacks.
3. Processes have private heaps, and threads have private stacks.
4. Processes share a heap, and theads share a stack.
5. Processes are for lawyers, and you have no idea of this stack-heap thing.

OpenMP

- A simple extension to sequential compilers to describe parallelism, based on:
  - Shared memory (threads),
  - Loops.
- Basic reference: http://www.openmp.org
- Designed by a consortium of companies
  - PGI, KAI, etc… (Compilers)
  - Intel, IBM, SGI, Sun, HP (HW + Compilers),
  - DoE, ASCI, NAG (users)
- You now get an OpenMP compiler directly "off the box".

A pragma-based approach

```
double res[10000];

for (i=0 ; i<10000 ; i++)
   compute(&res[i]) ;
```

A pragma-based approach

```
double res[10000];

for (i=0 ; i<10000 ; i++)
   compute(&res[i]) ;
```

```
#include "openmp.h"
double res[10000];
 #pragma omp parallel for
for (i=0 ; i<10000 ; i++)
   compute(&res[i]) ;
```

A simple flag at compile-time enables or disables the parallelism:

- gcc –fopenmp foo.c –o foo
- Export OMP_NUM_THREADS=4
- ./foo

Pragmas vs. Lib calls

In OpenMP, you find:

- Compile-time pragma
  - The compiler uses them to decide when to create threads, how to distribute the computations, etc.
  - Ex.: #pragma omp num_threads(4)
- Call to a runtime library
  - Executed at runtime!
  - Ex.: omp_set_num_threads(4)
  - Should be protected by #ifdef _OPENMP ... #endif

Basic OpenMP constructs

- #pragma openmp parallel
  - Creates the threads.
  - Can be integrated (or not) with other pragmas
- #pragma openmp sections
  - Declares sections of code to be run in different threads.

Example - sections

```
#pragma omp parallel
{    // now the threads are running
  #pragma omp sections
  foo1( );
  #pragma omp section
  bar2( );
  #pragma omp section
  foo2( );
}
```

Simple & efficient, yet...

- How do the threads deal with concurrent accesses in the memory?
- OpenMP enables the declaration of private vs. shared variables.
  - The pragma are complemented with shared(x), private (z), etc...
- Some variables are automatically private or shared:
  - Var. declared before the thread creation are shared.
  - Var allocated in the heap are shared (why)?
  - Local variables are private (why?)

One more example

```
double A[10000];
omp_set_num_threads(4);
#pragma omp parallel
{
   int th_id = omp_get_thread_num();
   compute(th_id, A);
}
 printf("Done.");
```

One more example

```
double A[10000];
omp_set_num_threads(4);
#pragma omp parallel
{
   int th_id = omp_get_thread_num();
   compute(th_id, A);
}
 printf("Done.");
```

th_id is private.

One more example

```
double A[10000];
int th_id;
omp_set_num_threads(4);
#pragma omp parallel
{
  th_id = omp_get_thread_num();
  compute(th_id, A);
}
 printf("Done.");
```

th_id is shared.