



Utilising CMT Systems

Darryl Gove
Sun Microsystems Inc.

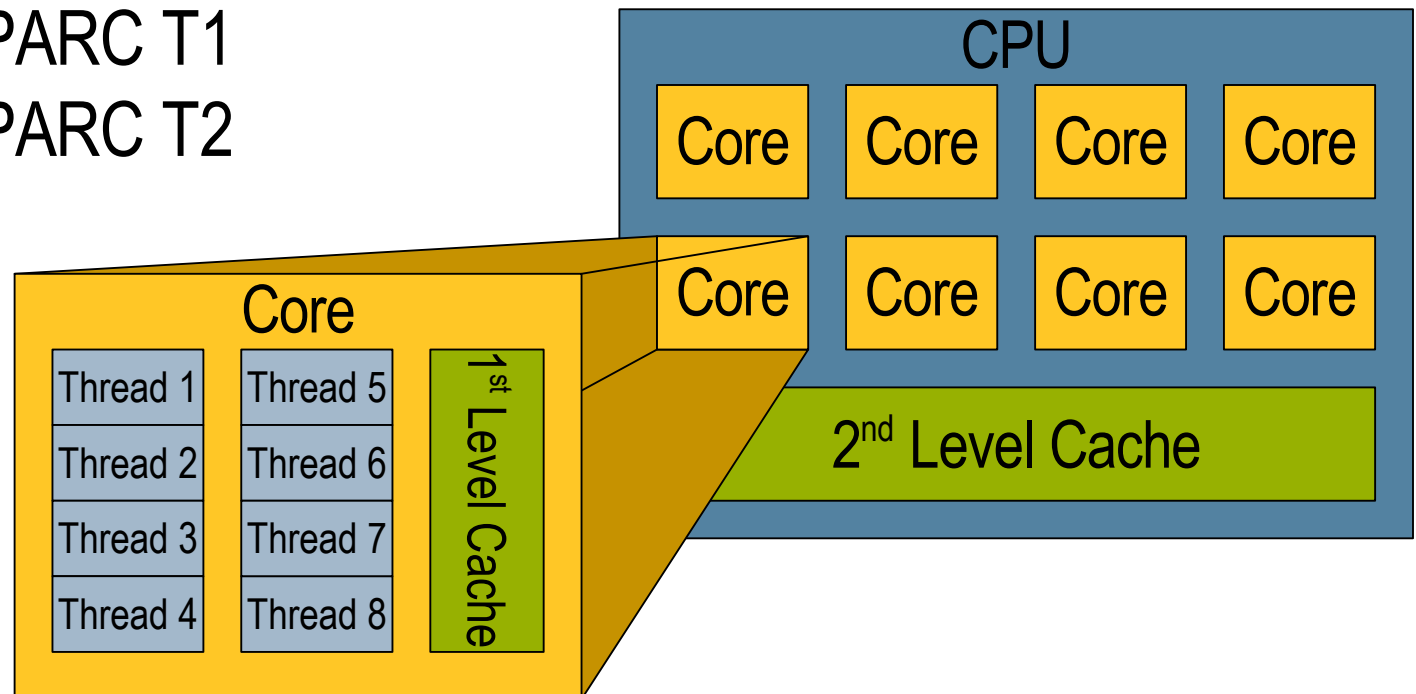


Outline

- What is CMT?
- Utilising CMT systems
- Developing for CMT systems
 - > OpenMP
 - > Automatic parallelisation
 - > Reductions
 - > Data races

Chip Multi-Threading (CMT)

- Supports many active threads simultaneously
- Not about the implementation details
- Examples:
 - > UltraSPARC T1
 - > UltraSPARC T2



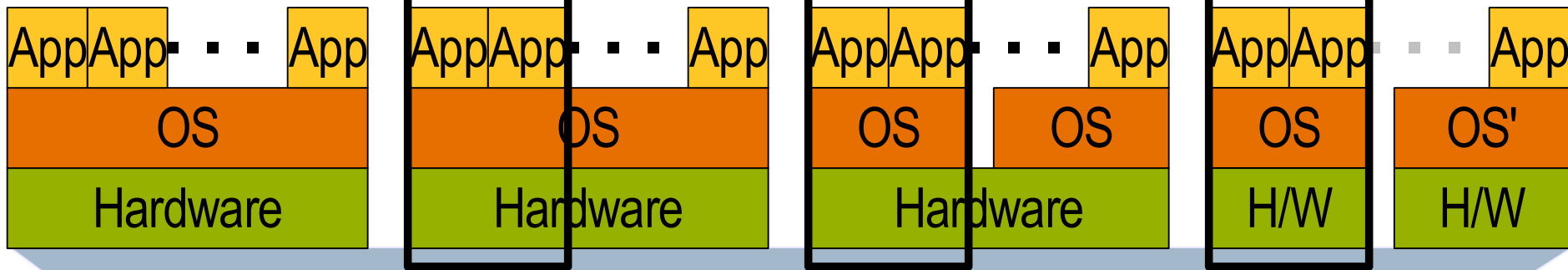
Utilising CMT Systems

Processes

Processor sets

Zones

Logical domains



Developing for CMT Systems

- Common options:
 - > POSIX threads (pthreads)
 - Very flexible
 - Can be hard
 - > OpenMP
 - Easy to use
 - Quite flexible
 - > Autopar
 - Trivial to use
 - Loop based

Using Autopar

```
void clear(int * array, int count)
{
    for (int i=0; i<count; i++)
    {
        array[i]=0;
    }
}
```

Output
parallelisation
information

```
% cc -c -O -xautopar -xvpara -xloopinfo clear.c
```

```
"clear.c", line 5: PARALLELIZED, and serial  
version generated
```

Enable automatic
parallelisation

Reductions

```
double total(double * array, int count)
{
    double ret=0;
    for (int i=0; i<count; i++)
    {
        ret += array[i];
    }
    return ret;
}
```

```
% cc -c -O -xautopar -xvpara -xloopinfo loop.c
"loop.c", line 4: not parallelized, unsafe
dependence (ret)
```

```
% cc -c -O -xautopar -xreduction -xvpara
-xloopinfo loop.c
```

```
"loop.c", line 4: PARALLELIZED, reduction, and
serial version generated
```

OpenMP

```
double sum(double * array, int count)
{
    double total =0;
    #pragma omp parallel for reduction(+:total)
    for (int i=0; i<count; i++)
    {
        total += array[i];
    }
    return total;
}
```

```
% cc -c -O -xopenmp -xvpara -xloopinfo l2.c
```

```
"l2.c", line 5: PARALLELIZED, user pragma used
```

Autoscopying

```
double sum(double * array, int count)
{
    double total =0;
    #pragma omp parallel for default(__auto)
    for (int i=0; i<count; i++)
    {
        total += array[i];
    }
    return total;
}
```

```
% cc -c -O -xopenmp -xvpara -xloopinfo l3.c
```

```
"l3.c", line 5: PARALLELIZED, user pragma used
```

Parallel sections

```
void set(int * a1, int * a2, int count)
{
    #pragma omp parallel sections
    {
        #pragma omp section
        {
            for (int i=0; i<count; i++)
            { a1[i]=0; }
        }
        #pragma omp section
        {
            for (int i=0; i<count; i++)
            { a2[i]=0; }
        }
    }
}
```

Tasks

```
while (ptr)
{
    #pragma omp task
    {
        process(ptr->data);
    }
    ptr=ptr->next;
}
```

OpenMP advantages

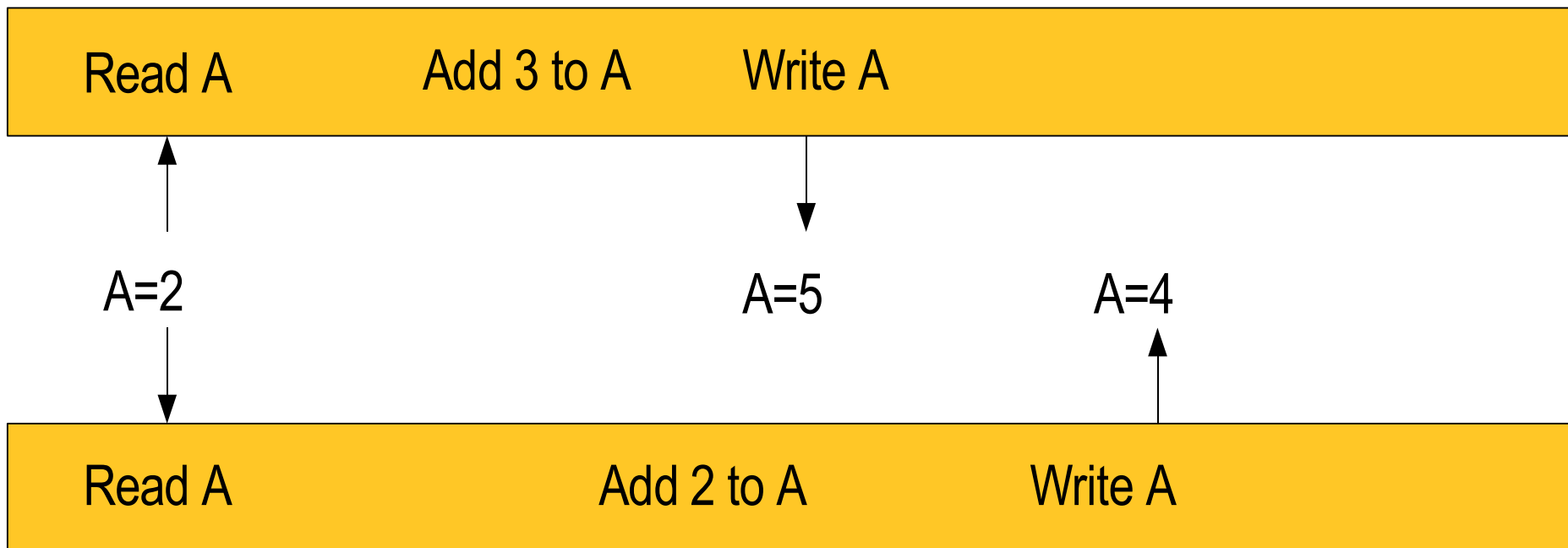
- Directive based
- Incremental
- Identical serial and parallel source

Effects of parallelisation

- Different threads can execute at different rates
- Variables need to be written back before another thread can see them
- Data shared between threads needs to be kept consistent

Data races

- Error in sharing of data between threads
- Can be detected using the Thread Analyzer



Parallelising a serial code

- Profile to identify hot spots
- Identify largest possible chunks of work
- Check for data races (& use compiler output)
- Check for performance gains
- Repeat

Summary

- Many ways to fully utilise a CMT system
- Parallel applications easy to develop using
 - > Automatic parallelisation
 - > OpenMP



Utilising CMT Systems

Darryl Gove

darryl.gove@sun.com

<http://blogs.sun.com/d/>

