Parallel Computing with GPUs

Advanced OpenMP
Part 2 - Scheduling

The University Of Sheffield.

Dr Paul Richmond
http://paulrichmond.shef.ac.uk/teaching/COM4521/

This Lecture (learning objectives)

- Scheduling
  - Compare and contrast different scheduling approaches to understand the benefits and limitations of each
  - Identify how scheduling parameters may impact cache utilisation

Scheduling clause

- OpenMP by default uses static scheduling
  - Static: schedule is determined at compile time
  - `schedule(static)`
- In general: `schedule(type [, chunk size])`
  - `type=static`: Iterations assigned to threads before execution (preferably at compile time)
  - *`type=dynamic`: iterations are assigned to threads as they become available*
  - *`type=guided`: iterations are assigned to threads as they become available (with reducing chunk size)*

  - `type=auto`: compiler and runtime determine the schedule
  - `type=runtime`: schedule is determined at runtime by env variable

What would be a use case where static scheduling is a bad choice?

Static scheduling chunk size

- chunk size
  - Refers to the amount of work assigned to each thread
  - By default chunk size is to divide the work by the number of threads
    - Low overhead (no going back for more work)
    - Not good for uneven workloads
    - E.g. consider our last lectures Taylor series example (updated to use reduction)

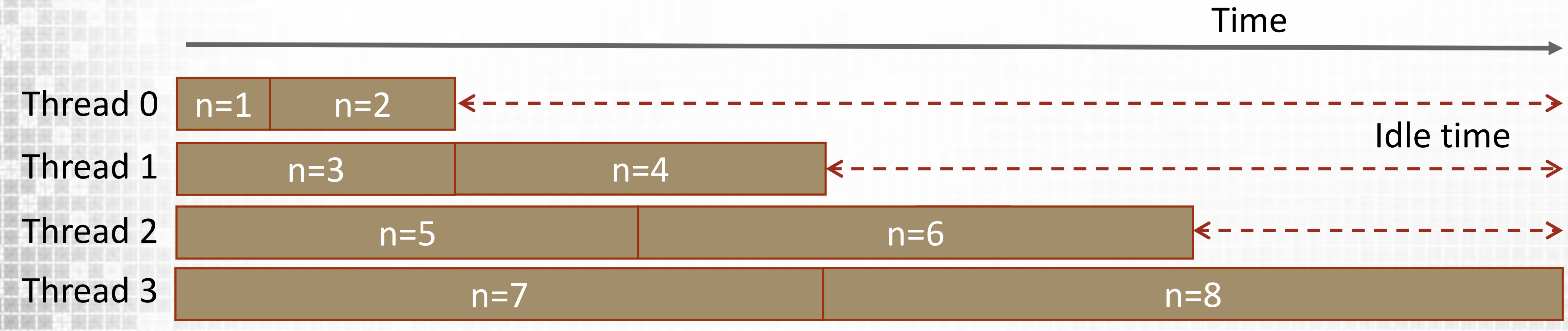
```c
int n;
double result = 0.0;
double x = 1.0;

#pragma omp parallel for reduction(-: result)
  for (n = 0; n < EXPANSION_STEPS; n++){
    double r = pow(-1, n - 1) * pow(x, 2 * n - 1) / fac(2 * n);
    result -= r;
  }

printf("Approximation is %f, value is %f\n", result, cos(x));
```

Recursive uneven workload

Scheduling Workload

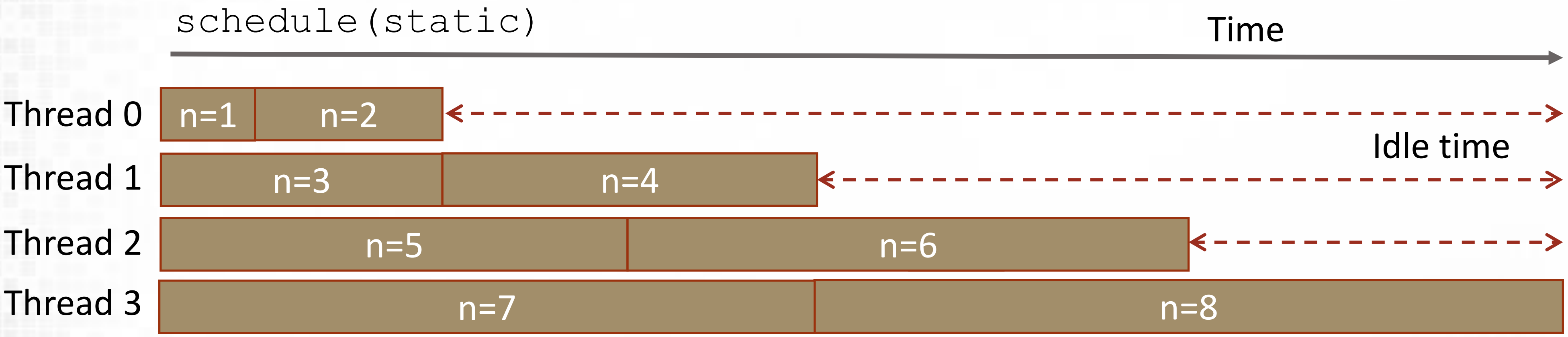
```
long long int fac(int n)
{
  if (n == 0)
    return 1;
  else
    return(n * fac(n - 1));
}
```

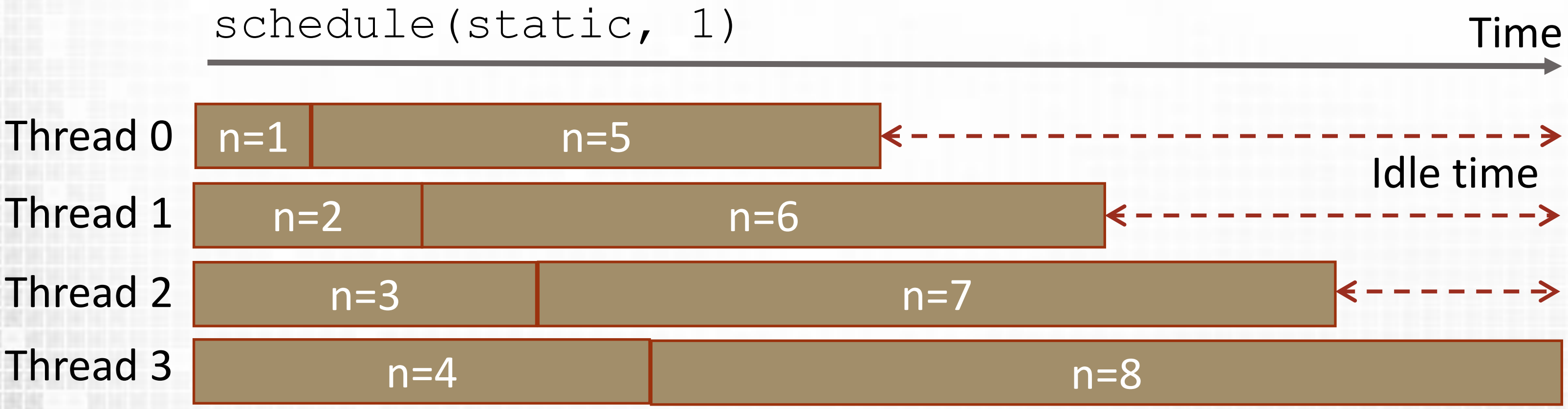
- Uneven workload amongst threads
  - Increase in *n* leads to increased computation
  - E.g. `EXPANSION_STEPS=8, num_threads(4), schedule(static)`



Cyclic Scheduling

- It would be better to partition the workload more evenly
  - E.g. Cyclic scheduling via chunk size





Cyclic Scheduling

```
#pragma omp for num_threads(4)
for (i = 0; i < 16; i++)
```

`schedule(static, 1)`

Thread 0	0	4	8	12
Thread 1	1	5	9	13
Thread 2	2	6	10	14
Thread 3	3	7	11	15

`schedule(static, 2)`

Thread 0	0	1	8	9
Thread 1	2	3	10	11
Thread 2	4	5	12	13
Thread 3	6	7	14	15

`schedule(static, 4)`

Thread 0	0	1	2	3
Thread 1	4	5	6	7
Thread 2	8	9	10	11
Thread 3	12	13	14	15

Default case

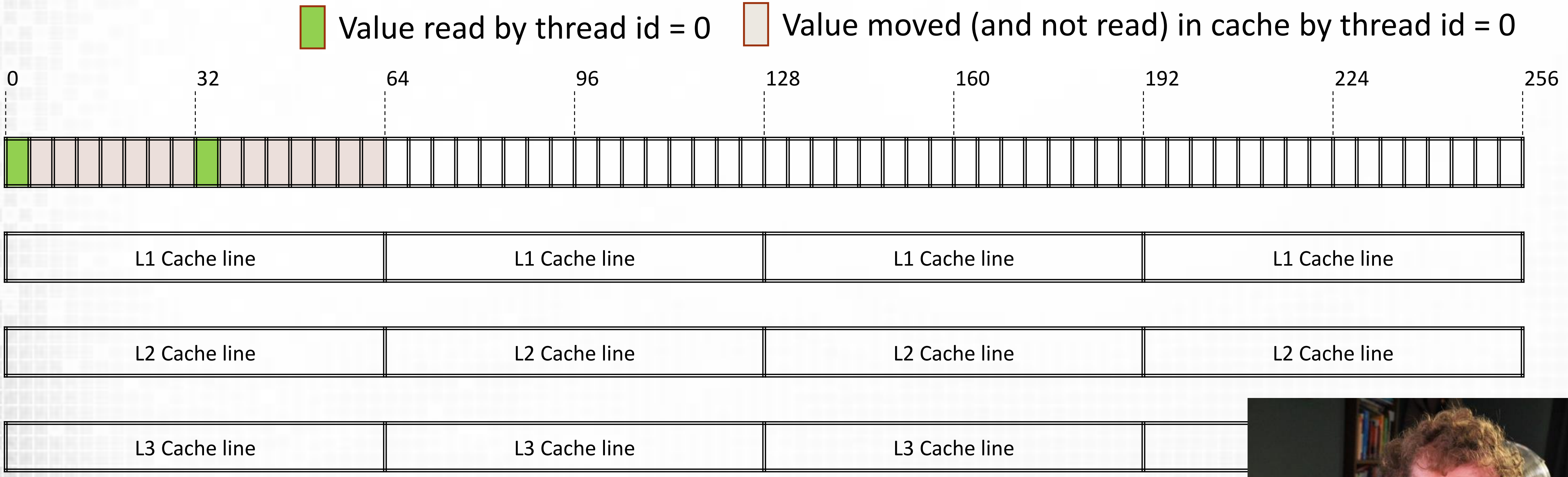
- Default chunk size is `n/threads`
  - where `n` is the number of iterations

Dynamic and Guided Scheduling

- Dynamic
  - **Iterations are broken down by chunk size**
  - Threads request chunks of work from a runtime queue when they are free
  - Default chunk size is 1
- Guided
  - **Chunks of the workload grow exponentially smaller**
  - Threads request chunks of work from a runtime queue when they are free
  - Chunk size is the size which the workloads decrease to
    - with the exception of last chunk which may have remainder
- Both
  - Requesting work dynamically creates overhead
    - Not well suited if iterations are balanced
  - Overhead vs. imbalance: How do I decide which is best?
    - **Benchmark all to find the best solution**

Cache Efficiency

```
#pragma omp parallel for schedule(static,1) num_threads(8)
    for (int i=0; i<64; i++) {
        something(array[i]);
    }
```

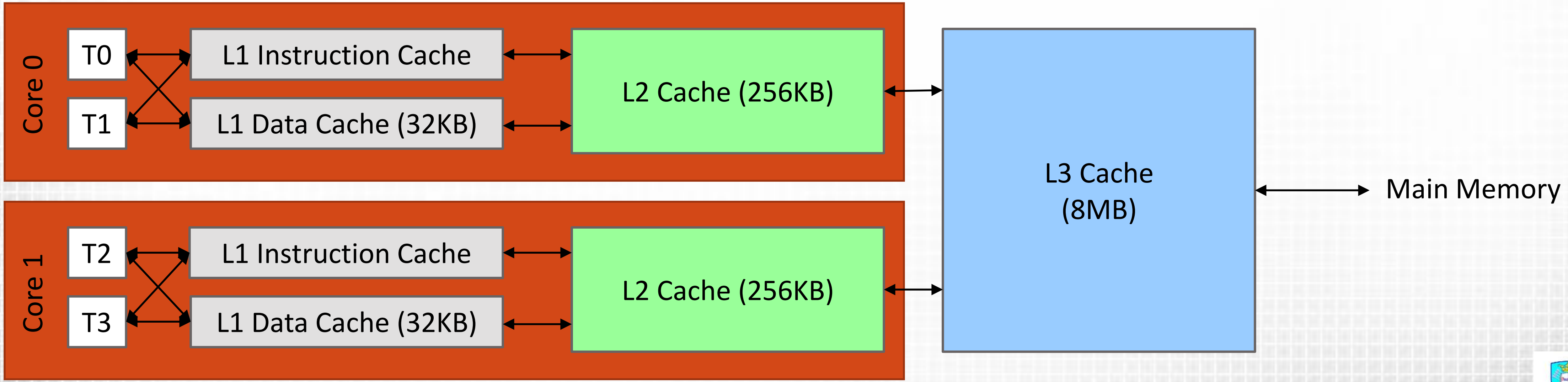


- ❑ Chunk size may effect cache utilisation

False Sharing

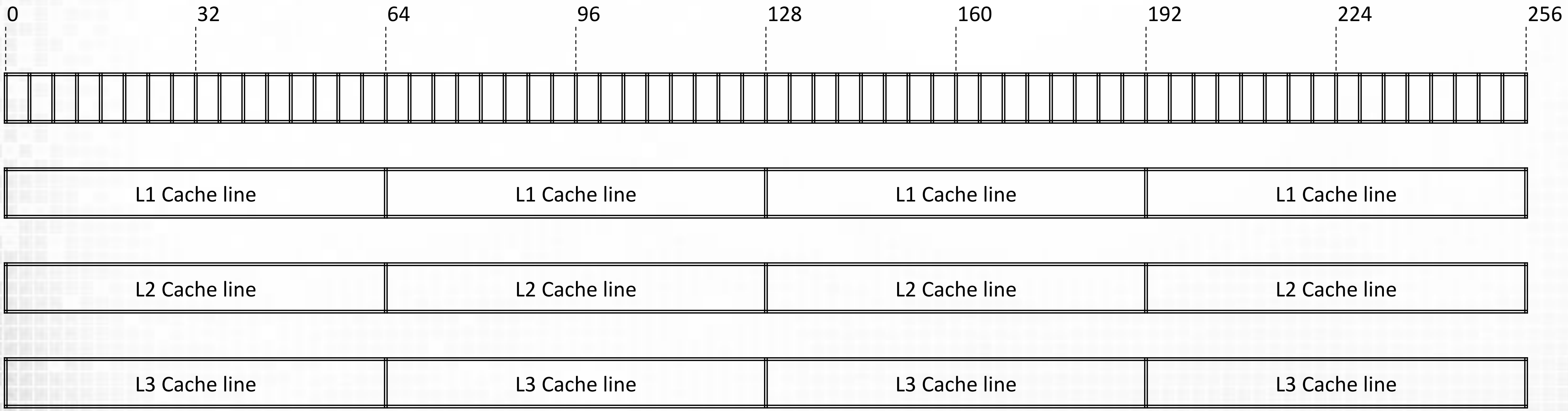
- Changing a single value causes a whole cache line to be invalid
  - Invalid lines must be re-cached
  - Chunk size case effect the amount of times a line is invalid

```
#pragma omp parallel for schedule(static,1)
    for (int i=0; i<64; i++) {
        array[i]++;
    }
```



False Sharing (worked example)

```
#pragma omp parallel for schedule(static,1) num_threads(8)
    for (int i=0; i<64; i++) {
        array[i]++;
    }
```



Summary

- ❑Scheduling
  - ❑Compare and contrast different scheduling approaches to understand the benefits and limitations of each
  - ❑Identify how scheduling parameters may impact cache utilisation

- ❑Next Lecture: Nesting Loops and OpenMP Summary