Contents

1 Dependencies

- A dependence is an ordering relationship **between two computations**. For correct results the ordering must be observed
- Examples:
  - One process waiting for data from another process
  - Two threads/processes reading/writing the same memory locations

Read more here

1.1 Key point

avoid introducing dependencies that do not matter to the computation - they will probably limit parallelism

2 Granularity

The granularity of a parallel computation is how much work (how many instructions) **can be done within a single thread** or process **between each interaction** with another thread or process

2.1 Fine-grained Parallelism

- Fine (small) grained parallelism has few instructions between interactions - **interaction is frequent**
- E.g. the code generated by a parallelising compiler from a sequential program
  - Each thread is likely to make **frequent access to shared variables**, perhaps doing only a few instructions between each access
  - OK on a multi-core machine with hardware shared memory because **interaction is relatively fast**

2.2 Course-grained Parallelism

- Coarse (large) grained parallelism has many instructions between interactions - **interaction is infrequent**
- E.g. shared processing projects (SETI@home, PI digitis calculation)
  - Each PC downloads enough work **for several hours or days**, uploaded only when complete
  - Very course-grained, because **interaction (over the internet) is slow**; Also makes use of the donors internet connection more efficient and time-limited

2.3 Reducing Granularity

- Batching is **performing work as a group**
- E.g. rather than sending each element of an array **individually** send all of the required elements **together**
- Or rather than a thread getting one small task at a time from a queue get several tasks at once
- Batching makes computation **more coarse-grained by reducing the frequency of interactions**
- But only makes sense if there are still **enough chunks of work for all the processors**, and the individual tasks dont have dependencies with other tasks

2.4 Increasing Granularity

- Over-dividing the work into **more, smaller, units** makes computation more **fine-grained**, since interaction is needed for (at least) every unit of work
- But can make it easier to keep all processors busy (e.g. with its own queue of several small jobs)
- Especially useful if units of work are **variable or unpredictable in size** since it is hard to divide the work evenly between processors

2.5 Key point

The granularity of parallelism must be appropriate for both the underlying hardwares resources and the solutions particular needs

3 Locality

3.1 Increasing locality by using More Memory

- Using extra memory can increase parallelism by reduced false dependencies (increasing locality)
- Privatisation: rather than threads competing to access a single shared variable, give each thread **its own separate copy** that can be used independently
- Padding: on some (shared memory) machines variables that are close together in memory **can be cached together**; adding extra padding can break this false dependency

3.2 Increasing locality through Redundant Computation

- In a redundant computation **each thread calculates the same value locally** - instead of calculating it once in one thread and communicating the value to each thread
- If each thread cannot make progress until it has the value then it may as well spend the time calculating it for itself
- The communication is no longer required

Reference section

placeholder