Reasoning About Performance

Chris Rossbach and Calvin Lin

"Solitude is independence"

-- Hermann Hesse, *Steppenwolf*

Outline

Sources of performance loss

Reasoning about performance

Performance tradeoffs

Ideal Performance Improvement

If it takes time T to solve a problem sequentially, we'd ideally like to solve it in time $\frac{T}{P}$ with *P* processors, giving us linear speedup

We typically do not obtain linear speedup

Sources of Performance Loss

1. Overhead
2. Non-parallelizable computation
3. Idle processors
4. Contention

Sources of Performance Loss (cont)

1. Overhead: Any cost that is not incurred by the sequential program

Examples

- Bookkeeping and other extra computation
- Communication and synchronization
- Controlling parallelism
- Extra memory usage

Sources of Performance Loss (cont)

2. Inherently sequential operations

Example

Traversal of a linked list

Sources of Performance Loss (cont)

3. Contention: Performance degradation that arises from competition for a shared resource

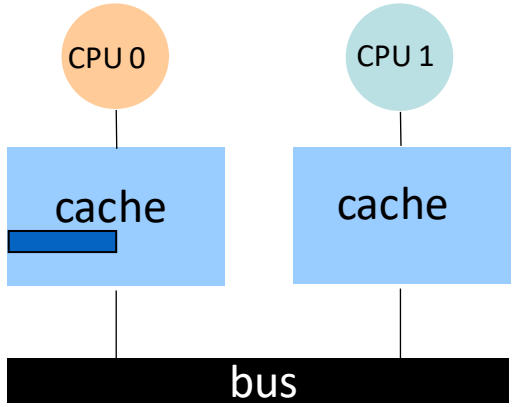
Examples

- Lock contention
- Bus contention
- False sharing

False Sharing

False Sharing

Occurs when non-shared objects reside on the same shared cache line



Example

CPU 0 repeatedly modifies address X

CPU 1 repeatedly modifies address X+1

If X and X+1 reside on the same cache line, that line will bounce between L0's cache and L1's cache

How can we avoid false sharing?

Pad the data to force non-shared objects to reside on different cache lines

Sources of Performance Loss (cont)

4. Idle Time

Causes

- Load imbalance
- Synchronization
- Slow processors

Discussion

What is the difference between contention and overhead?

Do we really need a separate category for contention?

Reasoning About Performance Loss

Let’s examine these causes of performance loss from different perspectives

Let’s phrase performance loss in terms of work

Performance Loss Phrased as Work

1. Overhead

   Extra work (bureaucracy)

2. Non-parallelizable computation

   Lack of parallel work

3. Idle processors

   Inability to work

4. Contention

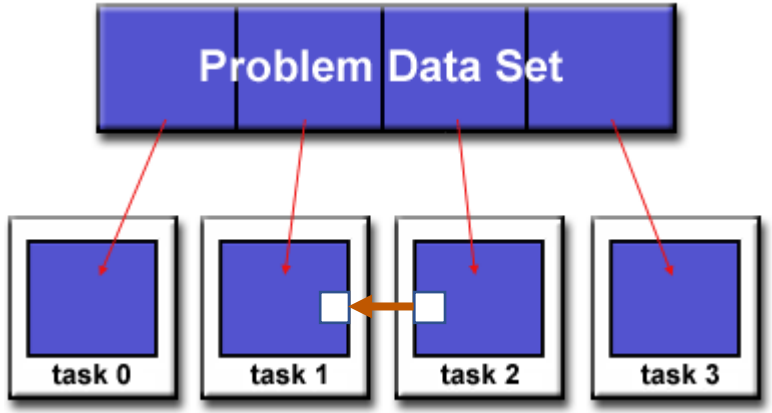
   Wasted work

Reasoning About Performance Loss

How do dependences help us reason about performance loss?

Independent work implies parallelism

Data dependences that cross threads imply the need to communicate or to synchronize




```
A[i] = A[i+1]
```

Can Performance Loss Always Be Traced to Dependences?

1. Overhead

   Extra work (bureaucracy)

2. Non-parallelizable computation

   Lack of parallel work

3. Idle processors

   Inability to work

4. Contention

   Wasted work

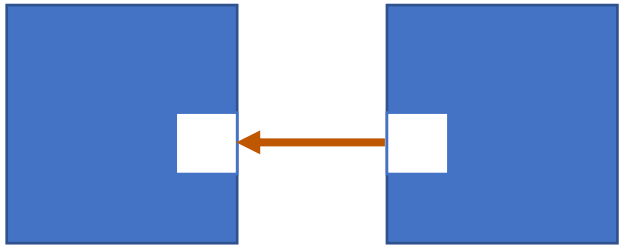
Not always, but dependences are helpful

Reasoning About Performance Loss

How do the notions of locality and granularity relate to performance?

Locality

Local operations avoid communication and synchronization

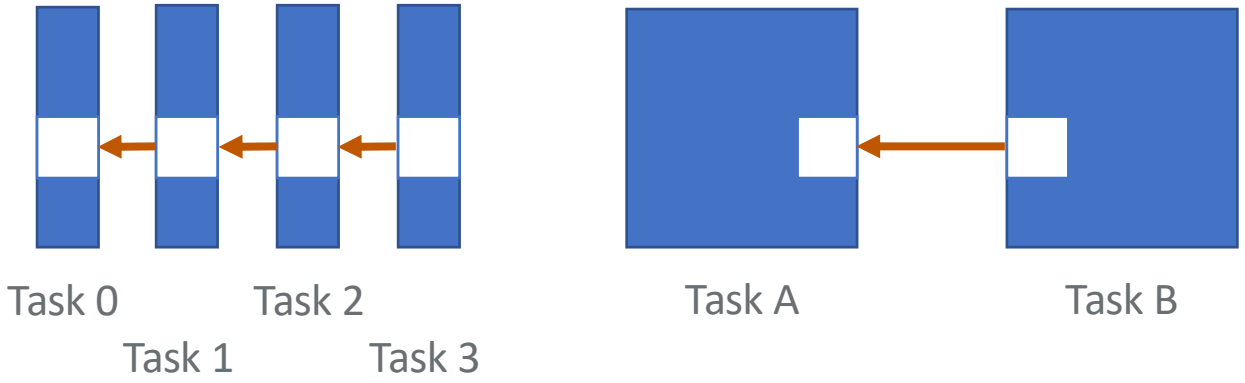


Reasoning About Performance Loss

How do the notions of locality and granularity relate to performance?

Granularity

Finer granularity tends to decrease locality and increase communication/synchronization



Corollary to Gustafson's Law

We can typically improve speedup by increasing the problem size

Why?

By increasing the problem size, we tend to increase the granularity (for the same number of processors)



Why is this true?

Geometric argument: For 2D problems, the surface area grows linearly while the area grows quadratically

Counter-examples?

Performance Optimizations

How can we improve performance/locality/granularity?

Computation vs. Communication

Overlap communication and computation

eg. Non-blocking operations: Hide latency, increase complexity

Perform redundant computation to avoid communication

eg. Compute the same random number on each processor instead of computing it once and sending it to all other processors

Performance Optimizations

How can we improve performance/locality/granularity?

Memory vs. Parallelism

Make extra copies of data to break dependences

eg. Array privatization

Pad data structures to avoid false sharing

Performance Optimizations

How can we improve performance/locality/granularity?

Granularity vs. Parallelism

Largest granularity possible?

Put all work in one thread! No overhead but no parallelism

Other examples?

Coarse-grained locks vs. fine-grained locks

Summary

There are many aspects of parallel performance

The notions of dependences, locality, and granularity are key

There are many possible tradeoffs to make