coursera

4.4 Lecture Summary

4 Dataflow Synchronization and Pipelining

4.4 Pipeline Parallelism

Lecture Summary: In this lecture, we studied how point-to-point synchronization can be used to build a one-dimensional *pipeline* with *p* tasks (stages), $T_0, \ldots, T_{p-1}$. For example, three important stages in a *medical imaging* pipeline are *denoising*, *registration*, and *segmentation*.

We performed a simplified analysis of the *WORK* and *SPAN* for pipeline parallelism as follows. Let *n* be the number of input items and *p* the number of stages in the pipeline, $WORK = n \times p$ is the total work that must be done for all data items, and $CPL = n + p - 1$ is the *span* or *critical path length* for the pipeline. Thus, the ideal parallelism is $PAR = WORK / CPL = np / (n + p - 1)$. This formula can be validated by considering a few boundary cases. When $p = 1$, the ideal parallelism degenerates to $PAR = 1$, which confirms that the computation is sequential when only one stage is available. Likewise, when $n = 1$, the ideal parallelism again degenerates to $PAR = 1$, which confirms that the computation is sequential when only one data item is available. When *n* is much larger than *p* ($n \gg p$), then the ideal parallelism approaches $PAR = p$ in the limit, which is the best possible case.

The synchronization required for pipeline parallelism can be implemented using phasers by allocating an array of phasers, such that phaser **ph[i]** is “signalled” in iteration i by a call to **ph[i].arrive()** as follows:

```
// Code for pipeline stage i
while ( there is an input to be processed ) {
  // wait for previous stage, if any
  if (i > 0) ph[i - 1].awaitAdvance();

  process input;

  // signal next stage
  ph[i].arrive();
}
```

Optional Reading:

1. Wikipedia article on Pipeline (computing).