Sources

- Section 4.4 of *Concurrent Programming in Java* (Doug Lea, Prentice Hall PTR, November 1999)
  - Covers parallel decomposition in greater detail.

- Section 6-7-8 of *Java concurrency in practice* (Brian Goetz, et al., Addison Wesley Professional May 09, 2006)

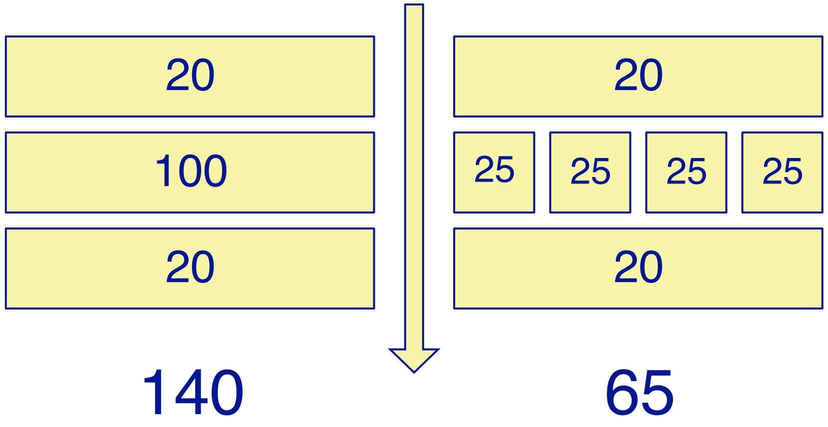
- Doug Lea's concurrency-interest website:
  - Download the fork-join framework as part of the jsr166y package
  - read the paper on its design.
  - http://gee.cs.oswego.edu/dl/concurrency-interest/index.html

Why we should practice parallel programming?

Because I want to keep my super cool multi-core computer busy!

Why we should practice parallel programming?

Amdahl's law



$$\text{Speed up} = \frac{1}{(1-p) + \dfrac{p}{n}}$$

p = part of parallel code

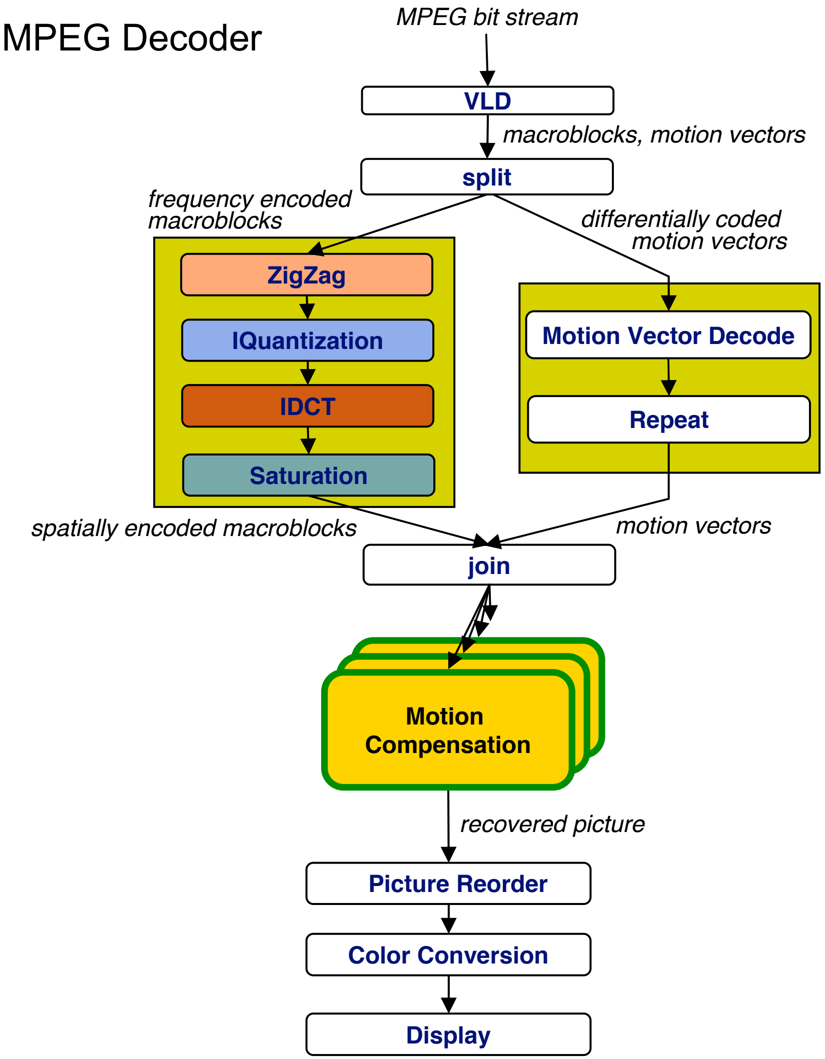
n = number of CPUs

Why we should practice parallel programming?

- Don’t try to force a non-parallel problem to be parallel
- Identify which are the program chunks that can provide the best ~~ratio~~ speedup/effort

Kinds of parallelism (Problem decomposition)

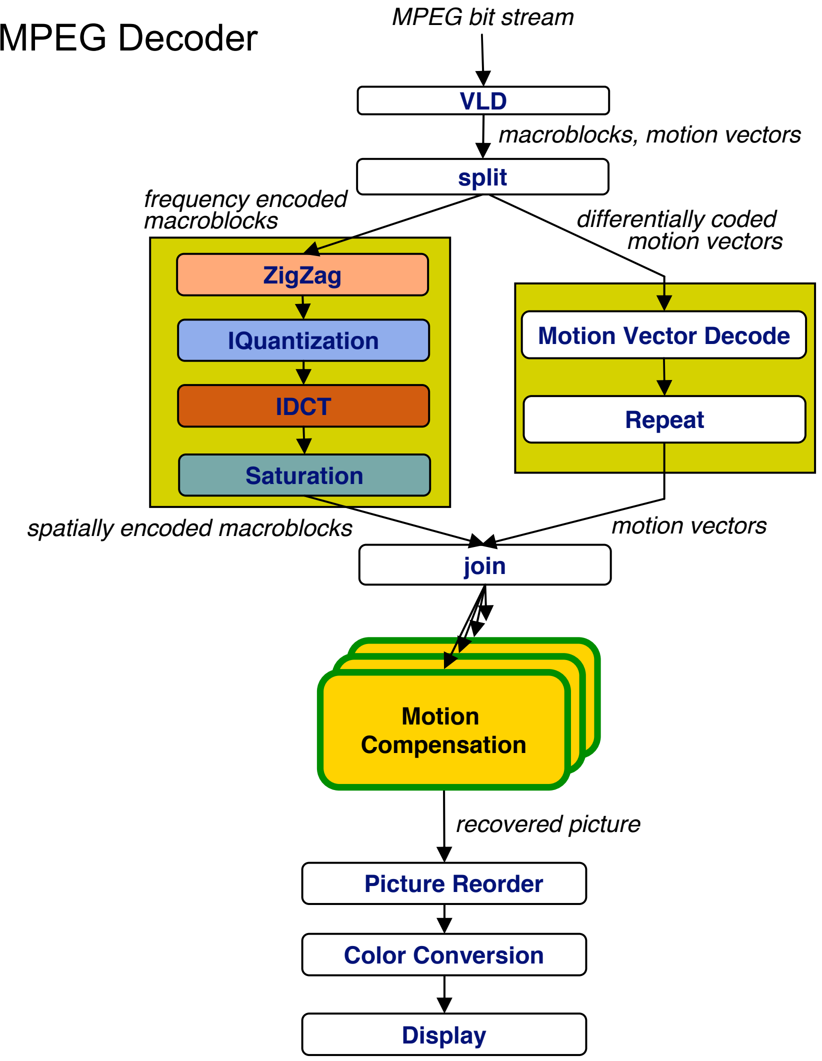
MPEG Decoder



Kind of parallelisms (Problem decomposition)

- **Data parallelism**: The same task run on different data in parallel
  - Can divide parts of the data between different tasks and perform the tasks in parallel
  - No dependencies among the tasks that cause their results to be ordered or merged

MPEG Decoder



Thread Pool

- A *thread pool* manages a set of worker threads.
- The threads into the pool have a simple life cycle:
  - Request the next task from the queue of tasks
  - Execute
  - And wait for another task

- Advantages from using a thread pool:
  - Reduce the costs of thread creation and teardown
  - Increases responsiveness
  - By properly tuning the pool you always have the correct number of threads (you don’t run out of memory and all your CPUs are busy)

Thread Pool

- newSingleThreadExecutor: A single-threaded executor creates a single worker thread to process tasks, replacing it if it dies unexpectedly. Tasks are guaranteed to be processed sequentially according to the order imposed by the task queue (FIFO, LIFO, priority order).

- newScheduledThreadPool: A fixed-size thread pool that supports delayed and periodic task execution, similar to Timer.

Merge sort

- *Divide et Impera* algorithm:
  - **Divide**: split your problem into sub-problems that are smaller parts of the original problem
  - **Impera**: solve the sub-problems recursively. (If the sub-problem is small enough than it is solve in a straightforward manner).
  - **Combine**: the solutions to the sub-problems into the solution for the original problem.