Slide 1

Administrivia

- A hint about project ideas/proposals: Try to think in terms of a basic/simple task you're pretty sure you can finish, plus some extras you can add as time permits.

Slide 2

Task Parallelism — Key Ideas

- Defining tasks: Might have some choices about what to regard as a task; make choices to give "enough tasks, but not too many."
- Dealing with "dependencies" (variables used by more than one task:
  - Removable dependencies (e.g., temporaries inside a loop) — replicate variable.
  - Separable dependencies (e.g., reductions) — replicate and combine at end.
  - Other dependencies (e.g., hidden ones in Monte Carlo homework) — see *Shared Data*.
- Scheduling tasks (assigning to UEs): Can do this statically (all decisions initially) or dynamically (e.g., master/worker scheme).

Slide 3

Task Parallelism — Examples

- Examples discussed earlier — numerical integration, molecular dynamics, Mandelbrot set (a.k.a. image construction).
- Monte Carlo homework problem also an example — hidden global variables for random number generation are "shared data" that must be managed.
- Probably the most widely applicable of these six patterns. So potentially lots and lots of project ideas. Might look for something that would use interesting scheduling, or where new tasks are created during execution, or where you can stop before completing all tasks.

Slide 4

Divide and Conquer — Key Ideas

- Informal pseudocode:

```
if (too small to parallelize)
    solve sequentially
else
    split into subproblems
    create task(s) for all but first subproblem
    solve first subproblem
    wait for created tasks to finish
    merge subsolutions
end if
```

- Amount of exploitable concurrency varies — at start and end, not much. So not necessarily a great strategy.

Slide 5

Divide and Conquer — Examples

- Examples discussed previously — mergesort.
- Other examples — almost any sequential divide-and-conquer algorithm, linear algebra examples mentioned in book.

Slide 6

Geometric Decomposition — Key Ideas

- Basic idea: Algorithm involves update (possibly repeated) of large data structure. Idea is to decompose into "chunks" and distribute, apply "owner computes" rule. Usually updating each chunk requires data in other chunks.
- Data decomposition (of key data structure): Choose to minimize communication. Arrays usually decomposed as in *Distributed Array*. Can simplify coding to include "ghost boundaries" to hold data from other chunks. Can replicate (or share) other data structures — e.g., reduction variable.
- Update operation: Each UE updates its "chunks". Must include communication to get data from other UEs if distributed memory, or synchronization (e.g., barriers) if shared memory.
- Data distribution (assigning chunks to UEs): Choose to minimize communication, balance computational load.

Slide 7

Geometric Decomposition — Examples

- Examples discussed previously — 1D heat distribution problem. Simple example of "mesh computation".
- Another example — matrix multiplication. See figures 4.17, 4.18, 4.20.
- Probably the second most widely applicable of these six patterns. So also lots of project ideas. Other mesh computations — iterative solvers for systems of linear equations, Conway's game of life, etc. Promising source of project ideas.

Slide 8

Recursive Data — Key Ideas and Examples

- Basic idea: Some operations on recursively-defined data structures (e.g., lists and trees) can be reworked to expose surprising concurrency.
- Concurrency usually very fine-grained, though, so more interesting for the ideas/thinking than for practical use.
- Examples previously discussed — finding roots in set of trees ("forest"). Other examples described/referenced in text. Possible source of project ideas, but again, more for ideas than for great performance (though it might be interesting to know how these do in OpenMP).

Slide 9

Pipeline — Key Ideas and Examples

- Basic idea: Assembly-line analogy. Informal pseudocode for each stage:

```
while (more data)
    receive data element from previous stage
    perform operation on data element
    send data element to next stage
end while
```

- Most natural implementation probably SPMD — one UE per pipeline stage, transferring data by message-passing or via queues shared between pipeline stages.
- Examples in book. Could be projects here, but might take some reading-up on applications.

Slide 10

Event-Based Coordination — Key Ideas and Examples

- Basic idea: Collection of semi-independent entities interacting irregularly. (Contrast with more regular interaction in *Pipeline*.) Model interaction in terms of "events" sent from one entity to another. Informal pseudocode for each stage:

```
while (not done)
    receive event
    process event
    send event(s)
end while
```

  Can be tricky if it events should be processed "in order" (e.g., by timestep) and might not arrive that way.
- Most natural implementation probably also SPMD — one UE per entity, sending events by message-passing or via queues shared among entities (e.g., one input queue for each entity).

Slide 11

- Example — car-wash simulation sketched in book. Pattern fits discrete-event simulation, and has been used for applications, though not widely — not trivial to get details right and get good performance. Could be projects here; could be interesting to find out how good performance is for reasonable programming effort.

Slide 12

Minute Essay

- None — sign in.