Slide 1

Administrivia

- Homework 1 sample solution code available (finally), for what it's worth.
- Reminder: Homework 2 due today. Okay to turn in tomorrow if you don't quite finish today.

Slide 2

Homework 2 Review and Questions

- One person asked about leapfrogging in the OpenCL version. (Discuss.)
- Other last-minute questions?

Slide 3

Finding Concurrency Design Space — Review/Recap

- First step is decompose problem into "tasks" that might be able to execute concurrently.
- Next step is to group tasks and figure out any ordering constraints.
- Next consider how tasks share data.
- Finally, review what we have so far and backtrack if need be.

Slide 4

Data Sharing

- Sometimes tasks are totally independent, each executes on totally separate data, etc. Usually not, though. Point here is to think through dependencies.
- Useful to think in terms of:
  - "Task-local" data — variables used only/mainly by single task, particularly the ones being updated. Example — chunks in heat diffusion problem.
  - Globally shared data — variables not associated with any particular task(s). Example — sum in numerical integration problem.
  - Data shared among smaller groups of tasks. Example — "boundary" points in heat diffusion problem.

Slide 5

Data Sharing, Continued

- Potential problems different in different environments; goal is to ensure correctness without adding too much overhead:
  - With shared memory, all UEs (can) have access to all data, but must use synchronization to prevent "race conditions".
  - With distributed memory, each UE has its own data, so race conditions not possible, but must use communication to (in effect) share data.
- Basic approach — first identify what data is shared, then figure out how it's used.

Slide 6

Data Sharing — Categories of Shared Data

- Read-only: Easiest case. If shared memory, don't need to do anything. If distributed memory, consider giving each process a copy. Examples include global constants.
- Effectively-local (large data structure, but each element accessed by only one UE): Also easy. If distributed memory, give each process "its" data.

Slide 7

Data Sharing — Categories of Shared Data, Continued

- Read-write (accessed by more than one task, at least one changing it): Can be arbitrarily complicated, but some common cases aren't too bad:
  - "Accumulate" (variable(s) used to accumulate result — usually a reduction). Example — sum in numerical integration problem. Give each task (or each UE) a copy and combine at end.
  - "Multiple-read/single-write" (multiple tasks need initial value, one task computes new value). Example — points near boundaries of chunks in heat diffusion problem. Create at least two copies, one for task that computes new value, other(s) to hold initial value for other tasks.

Slide 8

Molecular Dynamics Example — Analyze Task/Data Dependencies

- Arrays of atom positions, velocities:
  - Read-only for most groups of tasks — but tasks may need access to many elements, so for distributed memory might want to duplicate.
  - Updated by one group of tasks, but each task updates its own element(s) — "effectively local".
- Array of forces:
  - Read-only for group of tasks that update positions and velocities, and each task needs access only to "local" data.
  - Updated by several groups of tasks, but updates fit "accumulate data" model.

Slide 9

Molecular Dynamics Example — Task/Data Dependencies, Continued

- Array of neighbor lists:
  - Read-only for group of tasks that compute "non-bonded" forces, and each task needs access only to local data.
  - Updated by one group of tasks, but each task updates its own element(s).
- (Also see Figure 3.5 in book.)

Slide 10

Heat Diffusion Example — Analyze Task/Data Dependencies

- Arrays of old, new values:
  - Old values read-only for all groups of tasks, and each task needs access mostly to local data — plus "boundary values" for neighboring tasks.
  - New values updated by one group of tasks, and each task computes values only for "its" elements.

  For distributed memory, could distribute among processes, with extra variable(s) to hold copy of boundary values.
- Maximum difference between old, new values is "accumulate data" in one group of tasks, read-only elsewhere.
- Pointers to old/new values — changed at end of time step by one task, read-only elsewhere. Could duplicate for distributed memory.

Slide 11

Design Evaluation

- Idea of this pattern — questions to ask yourself about design/analysis before going further, to reduce odds of costly mistakes.
- Ideal design is easy to implement/maintain and produces a fast program suitable for target architecture. (But keep in mind old saying from engineering: "Good, fast, cheap: Pick any two.")

Slide 12

Design Evaluation — Suitability for Target Platform

- How many processing elements (PEs) are available? Need at least one task per PE, often want many more — unless we can easily get exactly one task per PE at runtime, with good load balance. ("Load balance"? what it sounds like, maybe — all PEs have about the same amount of work to do.)
- How are data structures shared among PEs? If there's a lot of shared data, or sharing is very "fine-grained", implementing for distributed memory will likely not be easy or fast.

Slide 13

Design Evaluation — Suitability for Target Platform, Continued

- How many UEs are available and how do they share data? Similar to previous questions, but in terms of UEs — with some architectures, can have multiple UEs per PE, e.g., to hide latency. For this to work, "context switching" must be fast, and problem must be able to take advantage of it.
- How does time spent doing computation compare to overhead of synchronization/communication, on target platform? May be a function of problem size relative to number of PEs/UEs.

Slide 14

Design Evaluation — Design Quality

- Is it flexible? Will it adapt well to a range of platforms (if appropriate), differing numbers of UEs/PEs, different problem sizes? Does it deal gracefully with "boundary cases"?
- Is it efficient? Can you get good load balance? Is overhead minimal? consider UE creation and scheduling, communication, and synchronization.
- Is it (paraphrasing Einstein) "as simple as possible, but not simpler"? Is it reasonable to think mortals can produce working code relatively quickly? which can later be ported and/or enhanced?

Slide 15

Design Evaluation — Preparation for Next Phase

- How regular are tasks and their data dependencies?
- Are interactions between tasks (or groups of tasks) synchronous or asynchronous?
- Are tasks grouped in the best way?

Slide 16

Molecular Dynamics Example — Design Evaluation

- Major phases of computation seem to involve a lot of tasks, so we can take advantage of many processors.
- Data sharing seems more suited to shared memory than distributed memory, but the latter could work if we just duplicate data (have to think about how well that would “scale”).
- Tasks and data are fairly regular, with one exception: how many neighbors an atom has might vary a lot. Probably will affect how we split up work among UEs.
- Interaction among tasks is synchronous.

Slide 17

Heat Diffusion Example — Design Evaluation

- Major phases of computation seem to potentially involve a lot of tasks, so we can take advantage of many processors.
- Data sharing seems suitable for either shared or distributed memory.
- Tasks and data are very regular, interaction is synchronous.

Slide 18

Minute Essay

- Anything interesting to report about Homework 2?