

Slide 1

Administrivia

- Homework 2 deadline extended again, to next Wednesday. Updates in next slide.
- Course “useful links” page has links to OpenCL documentation: OpenCL home page links to specification that talks about concepts and has a nice picture illustrating relationship among compute units, work items, and work groups. “Online man pages” is a reference describing all the library routines in detail.

Slide 2

Homework 2 Updates

- The Java program should get the number of threads from an additional command-line argument. I think that makes more sense than an environment variable, and I’ve updated the numerical integration example accordingly.
- The OpenCL program should also get — something about how many work items, work groups, etc., from command-line arguments. Details coming soon. This program may be difficult, so there may be more hints coming by e-mail or in the writeup.
- I’m not fussy about program names, but please stick to the writeup’s description of command-line arguments.

Slide 3

Minute Essay From Last Lecture

- Most people had started, but not all! but I strongly advise that you try to get everything but the OpenCL part done soon.

Slide 4

Numerical Integration in OpenCL, Revisited

- (Look at sample code one more time?)

Slide 5

A Few Words About Design Patterns

- Title of our book includes the word “patterns”.
- What do we mean? “Design patterns”.

Slide 6

A Few (More) Words About Design Patterns

- Idea originated with architect Christopher Alexander (first book 1977). Briefly — look for problems that have to be solved over and over, and try to come up with “expert” solution, write it in a form accessible to others. Usually this means adopting “pattern format” to use for all patterns. Characteristics of a good pattern:
 - Neat balancing of competing “forces” (tradeoffs).
 - Name either tells you what it’s about, or is a good addition to vocabulary.
 - “Aha!” aspect.
- First used in CS in OOD/OOP, about 1987. Really started to take off in OO community with “Gang of Four” book (Gamma, Helms, Johnson, and Vlissides; 1995). Now can find people writing patterns in many, many areas.
- Simple low-level example — iterator.

“A Pattern Language for Parallel Programming”?

Slide 7

- Goal of our book (and preceding work) — apply this idea in parallel computing.
- We started out looking for patterns representing high-level structures for parallel programs, thinking there might be a dozen of them.
- At some point we realized we also wanted to talk about how you get from the original problem to one of these structures — i.e., how do expert parallel programmers think about how to decompose a problem, etc.? and also about commonly-occurring data structures and program structures, and how to map high-level designs/structures into real programming environments.
- After much thought and discussion . . .

“A Pattern Language for Parallel Programming”, Continued

Slide 8

- Eventually — four-layer “pattern language”. (Notice that “pattern language” connotes common vocabulary more than grammatical structure. Not a programming language!)
- Much work has been done to revise and extend it, primarily by Mattson and Sanders and a group at UC Berkeley. Current status of this project — I don't know!

Overall Organization of Our Pattern Language

Slide 9

- Four “design spaces” corresponding to phases in design.
 - *Finding Concurrency* — how to decompose problems, analyze decomposition.
 - *Algorithm Structure* — high-level program structures.
 - *Supporting Structures* — program structures, data structures.
 - *Implementation Mechanisms* — generic discussion of programming environment “building blocks”.
- Idea is that you start at the top, work your way down, possibly with some backtracking.

Finding Concurrency — Preview

Slide 10

- Decomposition patterns (*Task Decomposition, Data Decomposition*): Break problem into tasks that maybe can execute concurrently.
- Dependency analysis patterns (*Group Tasks, Order Tasks, Data Sharing*): Organize tasks into groups, analyze dependencies among them.
- *Design Evaluation*: Review what you have so far, possibly backtrack.

Slide 11

Algorithm Structure — Preview

- *Task Parallelism* — decompose problem into lots of tasks, independent or nearly so. Example: numerical integration.
- *Divide and Conquer* — decompose recursively as in divide-and-conquer algorithms. Examples: quicksort, mergesort.
- *Geometric Decomposition* — decompose based on data (by rows, by columns, etc.). Example: Mesh-based computation.
- *Recursive Data* — rethink computation to expose unexpected concurrency. Ignore for now.
- *Pipeline* — decompose based on assembly-line analogy.
- *Event-Based Coordination* — decompose problem into entities interacting asynchronously.

Slide 12

Supporting Structures — Preview

- Program structure patterns:
 - *SPMD* (Single Program, Multiple Data) — “like an MPI program”.
 - *Loop Parallelism* — “like an OpenMP program”.
 - *Master/Worker* — like the name suggests.
 - *Fork/Join* — when none of the others fits.
- Data structure patterns:
 - *Shared Data* — generic advice for dealing with data dependencies.
 - *Shared Queue* — example of applying *Shared Data*.
 - *Distributed Array*.

Implementation Mechanisms — Preview

Slide 13

- Generic discussion of “building blocks” for parallel programming — analogous to assignment, if/then/else, loops in procedural programming languages. (Can think of this as “what basic questions do I ask about a new parallel programming environment?”)
- Three basic categories:
 - UE management.
 - Synchronization.
 - Communication.

Minute Essay

Slide 14

- Have you encountered design patterns previously? maybe in another course, or in other work? (Where?)