Slide 1

Administrivia

- Homework 1 due Friday (11:59pm). Submit code and timings by e-mail. Questions? Remember I have “open lab” this afternoon, and office hours tomorrow afternoon.
- Also notice that the “useful links” page is no longer blank.

Slide 2

More Background — A Few Words About Performance

- If the point is to “make the program run faster” — can we quantify that?
- Sure. Several ways to do that. One is “speedup” —

$$S(P) = \frac{T_{total}(1)}{T_{total}(P)}$$

- What would you guess is the best possible value for $S(P)$?

Slide 3

Amdahl's Law

- Of course, most "real programs" have some parts that have to be done sequentially. Gene Amdahl (principal architect of early IBM mainframe(s)) argued that this limits speedup — "Amdahl's Law":

  If γ is the "serial fraction", speedup on P processors is (at best — this ignores overhead)

  $$S(P) = \frac{1}{\gamma + \frac{1-\gamma}{P}}$$

  and as P increase, this approaches $\frac{1}{\gamma}$ — upper bound on speedup.

  (Details of math in chapter 2.)

Slide 4

Parallel Overhead

- As we will find out — many reasons why a "real" parallel program might be slower than Amdahl's Law predicts.
- For shared-memory programming — if we need to synchronize use of shared variables, that takes time.
- For message-passing programming — sending messages takes time. Typically time to send a message involves a fixed cost plus a per-byte cost.
- Also, "poor load balance" may slow things down.
- But sometimes we can speed things up by "overlapping computation and communication".

Slide 5

MPI — Recap

- Intended as a single standard for message-passing programs. Many implementations.
- Programs (at least in MPI 1.0) follow SPMD model — many processes, all running the same source code, but each with its own memory space and each with a different ID.
- Source code in C/C++/Fortran, with calls to MPI library functions.

Slide 6

MPI Library — Review

- Setup and bookkeeping — initialization, cleanup, environment query, etc.
- Data management — pack/unpack, derived data types.
- Point-to-point communication — several varieties, differing mostly in how much synchronization.
- Collective operations — e.g., broadcast.

Slide 7

Simple (Blocking) Point-to-Point Communication in MPI

- Send with MPI_Send — returns as soon as data has been copied to system buffer, buffer in program can be reused.
- Receive with MPI_Recv — waits until message has been received.
- Can use "tags" to distinguish between kinds of messages. Can receive selectively or not (MPI_ANY_TAG). Received tag is in returned MPI_Status variable (e.g., status.MPI_TAG).
- Can receive from specific sender or from any sender. (MPI_ANY_SOURCE). Sender is in returned MPI_Status variable (e.g., status.MPI_SOURCE).
- For MPI_Recv, "length" parameter specifies buffer length. Use MPI_Get_count to get actual count.

Slide 8

Not-So-Simple Point-to-Point Communication in MPI

- For not-too-long messages and when readability is more important than performance, MPI_Send and MPI_Recv are probably fine.
- If messages are long, however, buffering can be a problem, and can even lead to deadlock. Also, sometimes it's nice to be able to overlap computation and communication.
- Therefore, MPI offers several other kinds of send/receive functions, including:
  - Synchronous (MPI_Ssend, MPI_Recv) — blocks both sender and receiver until communication can occur.
  - Non-blocking send/receive (MPI_Isend, MPI_Irecv, MPI_Wait) — doesn't block, program must explicitly test/wait.
  - Which is faster/better? probably best to try them and find out. (Sample programs exchange*.)

Slide 9

Collective Communication in MPI

- "Collective communication" operation — one that involves many processes (typically all, or all in MPI "communicator").
- Could implement using point-to-point message passing, but some operations are common enough to be library functions — broadcast (`MPI_Bcast`), "reduction" (`MPI_Reduce`), etc.

Slide 10

Timing MPI Programs

- "How long did it take?" often of interest. Can use system tools (e.g., `time` command) to check total elapsed time. Or can time "interesting" parts of program:

  `MPI_Wtime` returns elapsed time; call twice and subtract to find out how long something takes (`time_msg.c` on "sample programs" page).
- How meaningful output is depends — e.g., on whether the system is otherwise idle. Probably best to repeat observations a few times, and do some sort of averaging.

Minute Essay

- None — sign in.

Slide 11