inst.eecs.berkeley.edu/~cs61c

UC Berkeley CS61C : Machine Structures

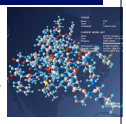
Lecture 42 – Software Parallel Computing

2007-05-02

Thanks to Prof. Demmel for his CS267 slides & Andy Carle for 1st CS61C draft
www.cs.berkeley.edu/~demmel/cs267_Spr05/

TA Matt Johnson

inst.eecs.berkeley.edu/~cs61c-tf



PS3 Folding at 600 TFLOPS!
Folding@home distributed computing
Playstation 3 now contributes 2/3rds of total power
Only accounts for 110K of the 2M CPUs in project
http://fah-web.stanford.edu/cgi-bin/main.py?qtype=osstats

Today's Outline

- Motivation for Parallelism
- Software-Managed Parallelism Idea
  - Software vs. hardware parallelism
  - Problems SW parallelism can solve
  - Fundamental issues
- Programming for computer clusters
  - The lower-level model – Message Passing Interface (MPI)
  - Abstractions – MapReduce
  - Programming Challenges

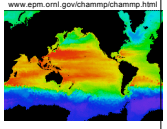
Big Problems

- Simulation: the Third Pillar of Science
  - Traditionally perform experiments or build systems
  - Limitations to standard approach:
    - Too difficult – build large wind tunnels
    - Too expensive – build disposable jet
    - Too slow – wait for climate or galactic evolution
    - Too dangerous – weapons, drug design
  - Computational Science:
    - Simulate the phenomenon on computers
    - Based on physical laws and efficient numerical methods
- Search engines needs to build an index for the entire Internet
- Pixar needs to render movies

Performance Requirements

- Performance terminology
  - the FLOP: FLoating point OPeration
  - Computing power in FLOPS (FLOP per Second)
- Example: Global Climate Modeling
  - Divide the world into a grid (e.g. 10 km spacing)
  - Solve fluid dynamics equations for each point & minute
    - Requires about 100 Flops per grid point per minute
  - Weather Prediction (7 days in 24 hours):
    - 56 Gflops
  - Climate Prediction (50 years in 30 days):
    - 4.8 Tflops

www.epm.ornl.gov/chammp/chammp.html

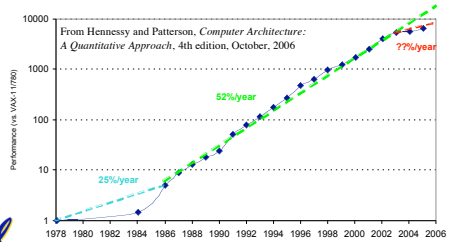


- Perspective
  - Pentium 4 3GHz Desktop Processor
    - ~6-12 Gflops
    - Climate Prediction would take ~50-100 years

Reference:http://www.hpcwire.com/hpcwire/hpcwireWWW/04/0827/108259.html

What Can We Do?

- Wait for our machines to get faster?
  - Moore's law tells us things are getting better; why not stall for the moment?
- Moore on last legs!
  - Many believe so … thus push for multi-core (Friday)!



Let's Put Many CPUs Together!

- Distributed computing (SW parallelism)
  - Many separate computers (each with independent CPU, RAM, HD, NIC) that communicate through a network
  - Grids (home computers across Internet) and Clusters (all in one room)
  - Can be "commodity" clusters, 100K+ nodes
  - About being able to solve "big" problems, not "small" problems faster
- Multiprocessing (HW parallelism)
  - Multiple processors "all in one box" that often communicate through shared memory
  - Includes multicore (many new CPUs)

The Future of Parallelism

"Parallelism is the biggest challenge since high level programming languages. It's the biggest thing in 50 years because industry is betting its future that parallel programming will be useful."

– David Patterson

Administrivia

- Dan's OH moved to 3pm Friday
- Performance competition submissions due May 8th
  - No slip days can be used!
- The final is Sat 5/12 12:30-3:30pm
  - Review session on Weds 5/9 at 2pm

Distributed Computing Themes

- Let's network many disparate machines into one compute cluster
- These could all be the same (easier) or very different machines (harder)
- Common themes
  - "Dispatcher" gives jobs & collects results
  - "Workers" (get, process, return) until done
- Examples
  - SETI@Home, BOINC, Render farms
  - Google clusters running MapReduce

Distributed Computing Challenges

- Communication is fundamental difficulty
  - Distributing data, updating shared resource, communicating results
  - Machines have separate memories, so no usual inter-process communication – need network
  - Introduces inefficiencies: overhead, waiting, etc.
- Need to parallelize algorithms
  - Must look at problems from parallel standpoint
  - Tightly coupled problems require frequent communication (more of the slow part!)
  - We want to decouple the problem
    - Increase data locality
    - Balance the workload

Programming Models: What is MPI?

- Message Passing Interface (MPI)

  - World's most popular distributed API
  - MPI is "de facto standard" in sci computing
  - C and FORTRAN, ver. 2 in 1997
  - What is MPI good for?
    - Abstracts away common network communications
    - Allows lots of control without bookkeeping
    - Freedom and flexibility come with complexity
      - 300 subroutines, but serious programs with fewer than 10
  - Basics:
    - One executable run on every node
    - Each node process has a `rank` ID number assigned
    - Call API functions to send messages

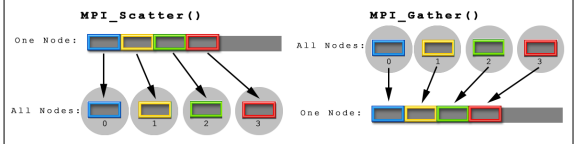
http://www.mpi-forum.org/
http://forum.stanford.edu/events/2007/plenary/slides/Olukotun.ppt
http://www.tbray.org/ongoing/When/200x/2006/05/24/On-Grids

Basic MPI Functions (1)

- `MPI_Send()` and `MPI_Receive()`
  - Basic API calls to send and receive data point-to-point based on `rank` (the runtime node ID #)
  - We don't have to worry about networking details
  - A few are available: blocking and non-blocking
- `MPI_Broadcast()`
  - One-to-many communication of data
  - Everyone calls: one sends, others block to receive
- `MPI_Barrier()`
  - Blocks when called, waits for everyone to call (arrive at some determined point in the code)
  - Synchronization

Basic MPI Functions (2)

- `MPI_Scatter()`
  - Partitions an array that exists on a single node
  - Distributes partitions to other nodes in rank order
- `MPI_Gather()`
  - Collects array pieces back to single node (in order)



Basic MPI Functions (3)

- `MPI_Reduce()`
  - Perform a "reduction operation" across nodes to yield a value on a single node
  - Similar to accumulate in Scheme
    - (accumulate + '(1 2 3 4 5))
  - MPI can be clever about the reduction
  - Pre-defined reduction operations, or make your own (and abstract datatypes)
    - `MPI_Op_create()`
- `MPI_AllToAll()`
  - Update shared data resource

MPI Program Template

- Communicators - set up node groups
- Startup/Shutdown Functions
  - Set up `rank` and `size`, pass `argc` and `argv`
- "Real" code segment

```
main(int argc, char *argv[]){
 MPI_Init (&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
 MPI_Comm_size(MPI_COMM_WORLD, &size);
 /* Data distribution */ ...
 /* Computation & Communication*/ ...
 /* Result gathering */ ...
 MPI_Finalize();
 }
```


Challenges with MPI

- Deadlock is possible…
  - Seen in CS61A – state of no progress
  - Blocking communication can cause deadlock
    - "crossed" calls when trading information
    - example:
      - Proc1: `MPI_Receive(Proc2, A); MPI_Send(Proc2, B);`
      - Proc2: `MPI_Receive(Proc1, B); MPI_Send(Proc1, A);`
    - There are some solutions - `MPI_SendRecv()`
- Large overhead from comm. mismanagement
  - Time spent blocking is wasted cycles
  - Can overlap computation with non-blocking comm.
- Load imbalance is possible…
- Things are starting to look hard to code!

A New Hope: Google's MapReduce

- Remember CS61A?

```
(reduce + (map square '(1 2 3)) ⇒
(reduce + '(1 4 9)) ⇒
14
```

- We told you "the beauty of pure functional programming is that it's easily parallelizable"
  - Do you see how you could parallelize this?
  - What if the `reduce` function argument were associative, would that help?
- Imagine 10,000 machines ready to help you compute anything you could cast as a MapReduce problem!
  - This is the abstraction Google is famous for authoring (but their reduce not the same as the CS61A's or MPI's reduce)
    - Builds a reverse-lookup table
  - It hides LOTS of difficulty of writing parallel code!
  - The system takes care of load balancing, dead machines, etc.

MapReduce Programming Model

Input & Output: each a set of key/value pairs

Programmer specifies two functions:

```
map (in_key, in_value) →
    list(out_key, intermediate_value)
```

- Processes input key/value pair
- Produces set of intermediate pairs

```
reduce (out_key, list(intermediate_value)) →
       list(out_value)
```

- Combines all intermediate values for a particular key
- Produces a set of merged output values (usu just one)

code.google.com/edu/parallel/mapreduce-tutorial.html

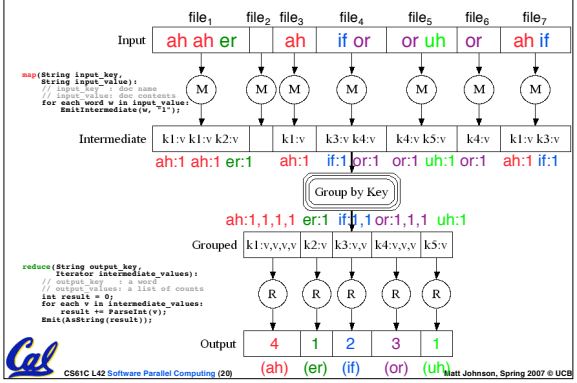
MapReduce Code Example

```
map(String input_key,
    String input_value):
    // input_key  : document name
    // input_value: document contents
    for each word w in input_value:
        EmitIntermediate(w, "1");

reduce(String output_key,
       Iterator intermediate_values):
    // output_key    : a word
    // output_values: a list of counts
    int result = 0;
    for each v in intermediate_values:
        result += ParseInt(v);
    Emit(AsString(result));
```

- "Mapper" nodes are responsible for the `map` function
- "Reducer" nodes are responsible for the `reduce` function
- Data on a distributed file system (DFS)

MapReduce Example Diagram

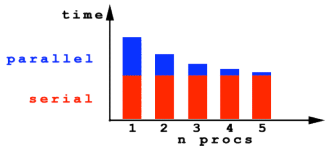


MapReduce Advantages/Disadvantages

- Now it's easy to program for many CPUs
  - Communication management effectively gone
    - I/O scheduling done for us
  - Fault tolerance, monitoring
    - machine failures, suddenly-slow machines, other issues are handled
  - Can be much easier to design and program!
- But… it further restricts solvable problems
  - Might be hard to express some problems in a MapReduce framework
  - Data parallelism is key
    - Need to be able to break up a problem by data chunks
  - MapReduce is closed-source – Hadoop!

Things to Worry About: Parallelizing Code

- Applications can almost never be completely parallelized; some serial code remains



- s is serial fraction of program, P is # of processors
- Amdahl's law:

Speedup(P) = Time(1) / Time(P)

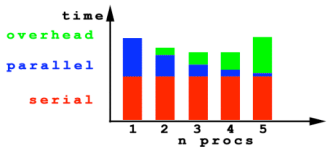
≤ 1 / (s + ((1-s) / P)), and as P → ∞

≤ 1/s

- Even if the parallel portion of your application speeds up perfectly, your performance may be limited by the sequential portion

But… What About Overhead?

- Amdahl's law ignores overhead
  - E.g. from communication, synchronization
- Amdahl's is useful for bounding a program's speedup, but cannot predict speedup



Summary

- Parallelism is necessary
  - It looks like the future of computing…
  - It is unlikely that serial computing will ever catch up with parallel computing
- Software parallelism
  - Grids and clusters, networked computers
  - Two common ways to program:
    - Message Passing Interface (lower level)
    - MapReduce (higher level, more constrained)
- Parallelism is often difficult
  - Speedup is limited by serial portion of code and communication overhead

To Learn More…

- About MPI…
  - www.mpi-forum.org
  - *Parallel Programming in C with MPI and OpenMP* by Michael J. Quinn
- About MapReduce…
  - code.google.com/edu/parallel/mapreduce-tutorial.html
  - labs.google.com/papers/mapreduce.html
  - lucene.apache.org/hadoop/index.html
- Try the lab, and come talk to me!

Bonus slides

- These are extra slides that used to be included in lecture notes, but have been moved to this, the "bonus" area to serve as a supplement.
- The slides will appear in the order they would have in the normal presentation

Bonus

How Do I Experiment w/ SW Parallelism?

1. Work for Google. ;-)
2. Use open source MapReduce and VMWare
   - Hadoop: map/reduce & distributed file system: `lucene.apache.org/hadoop/`
   - Nutch: crawler, parsers, index : `lucene.apache.org/nutch/`
   - Lucene Java: text search engine library: `lucene.apache.org/java/docs/`
3. Wait until tomorrow!!! (lab)
   - Google is donating a cluster for instruction!
     - Will be built for Fall 2007
   - We're developing MapReduce bindings for Scheme for CS61A
   - New MPI lab for CS61C
   - It will be available to students & researchers

Example Applications

- Science
  - Global climate modeling
  - Biology: genomics; protein folding; drug design; malaria simulations
  - Astrophysical modeling
  - Computational Chemistry, Material Sciences and Nanosciences
  - SETI@Home : Search for Extra-Terrestrial Intelligence
- Engineering
  - Semiconductor design
  - Earthquake and structural modeling
  - Fluid dynamics (airplane design)
  - Combustion (engine design)
  - Crash simulation
  - Computational Game Theory (e.g., Chess Databases)
- Business
  - Rendering computer graphic imagery (CGI), ala Pixar and ILM
  - Financial and economic modeling
  - Transaction processing, web services and search engines
- Defense
  - Nuclear weapons -- test by simulations
  - Cryptography