Lecturer SOE
Dan Garcia

inst.eecs.berkeley.edu/~cs61c

UCB CS61C : Machine Structures

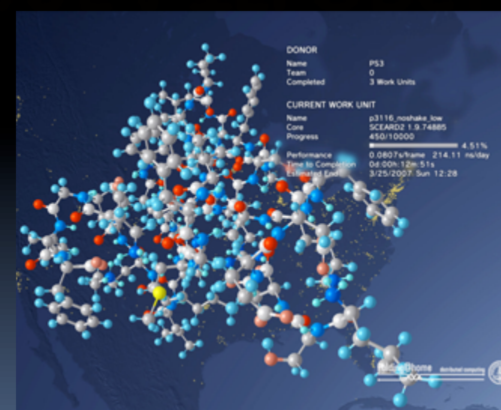
Lecture 37 – Inter-machine Parallelism

2010-04-26

Thanks to Prof. Demmel for his CS267 slides; and Andy Carle & Matt Johnson for CS61C drafts

GPU PROTEIN FOLDING UP TO 3954 TFLOPS!

Folding@home distributed computing says GPUs now contribute 66% of total performance (~4K/~6K x86 TFLOPS) but only 6% (~.3M/~5M) of "CPUs"!



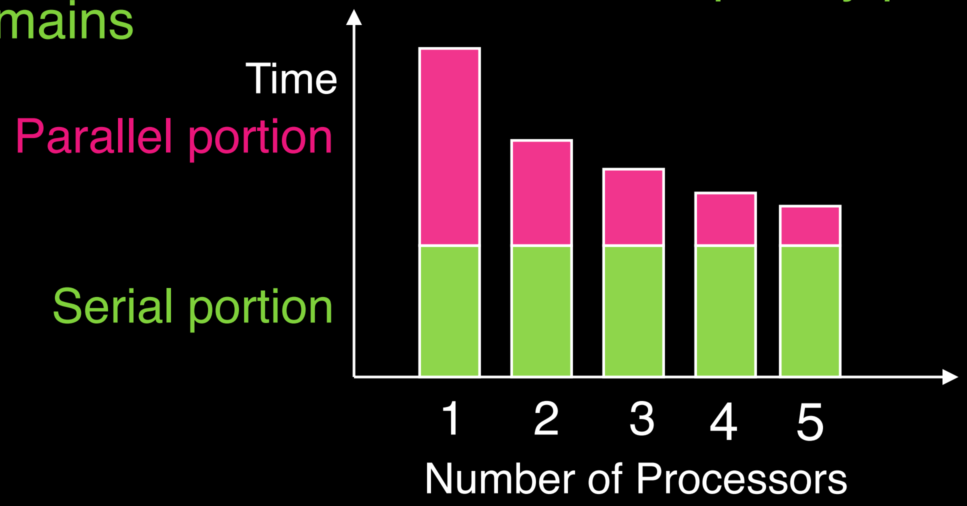
http://fah-web.stanford.edu/cgi-bin/main.py?qtype=osstats

Today's Outline

- **Amdahl's Law**
- **Motivation for Inter-machine Parallelism**
- **Inter-machine parallelism hardware**
  - Supercomputing
  - Distributed computing
    - Grid computing
    - Cluster computing
- **Inter-machine parallelism examples**
  - Message Passing Interface (MPI)
  - Google's MapReduce paradigm
  - Programming Challenges

Speedup Issues: Amdahl's Law

- Applications can almost <u>never</u> 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)

$\leq 1 / (s + [(1-s) / P)]$, and as $P \rightarrow \infty$

$\leq 1 / s$

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

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

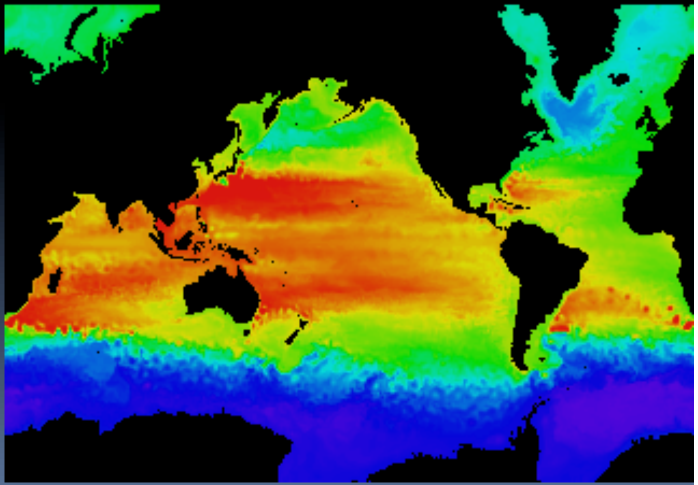
Example Applications

- **Science & Medicine**
  - 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

Performance Requirements

- **Performance terminology**
  - the FLOP: FLoating point OPeration
  - "flops" = # FLOP/second is the standard metric for computing power
- **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
- **Perspective**
  - Pentium 4 3GHz Desktop Processor
    - ~10 Gflops
    - Climate Prediction would take ~50-100 years

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

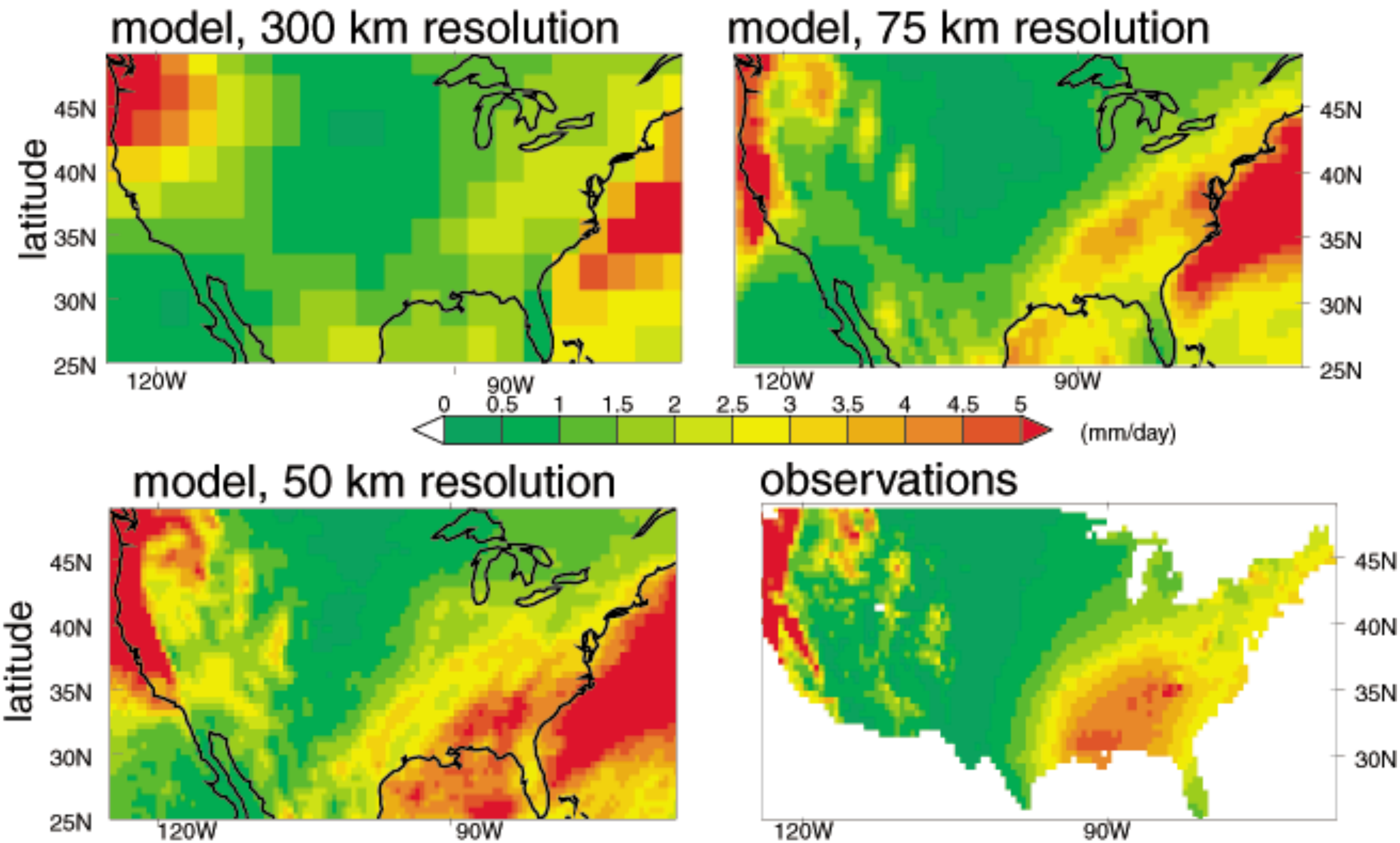


Reference:http://www.hpcwire.com/hpcwire/hpcwireWWW/04/0827/108259.html

High Resolution Climate Modeling on NERSC-3
P. Duffy, et al., LLNL

Wintertime Precipitation

As model resolution becomes finer, results converge towards observations



What Can We Do? Use Many CPUs!

- Supercomputing – like those listed in `top500.org`
  - Multiple processors "all in one box / room" from one vendor that often communicate through shared memory
  - This is often where you find exotic architectures
- Distributed computing
  - Many separate computers (each with independent CPU, RAM, HD, NIC) that communicate through a network
    - Grids (heterogenous computers across Internet)
    - Clusters (mostly homogeneous computers all in one room)
      - Google uses commodity computers to exploit "knee in curve" price/performance sweet spot
  - It's about being able to solve "big" problems, not "small" problems faster
    - These problems can be data (mostly) or CPU intensive

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 scientific 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

Challenges with MPI

- **Deadlock is possible…**
  - Seen in CS61A – state of no progress
  - Blocking communication can cause deadlock
- **Large overhead from comm. mismanagement**
  - Time spent blocking is wasted cycles
  - Can overlap computation with non-blocking comm.
- **Load imbalance is possible! Dead machines?**
- **Things are starting to look hard to code!**

Administrivia

- ??

Upcoming Calendar

Week #	Mon	Wed	Thu Lab	Fri
#14 Last week o' classes	Inter-machine Parallelism	Summary, Review, Evaluation	Parallel	Intra-machine Parallelism (Scott) P3 due
#15 RRR Week				Perf comp due 11:59pm
#16 Finals Week Review Sun May 9 3-6pm 10 Evans				Final Exam 8-11am in Hearst Gym

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`)
    - Often, their `reduce` builds a reverse-lookup table for easy query
  - 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 WordCount Example

- "Mapper" nodes are responsible for the `map` function

```
// "I do I learn" → ("I",1), ("do",1), ("I",1), ("learn",1)
map(String input_key,
    String input_value):
    // input_key  : document name (or line of text)
    // input_value: document contents
    for each word w in input_value:
        EmitIntermediate(w, "1");
```

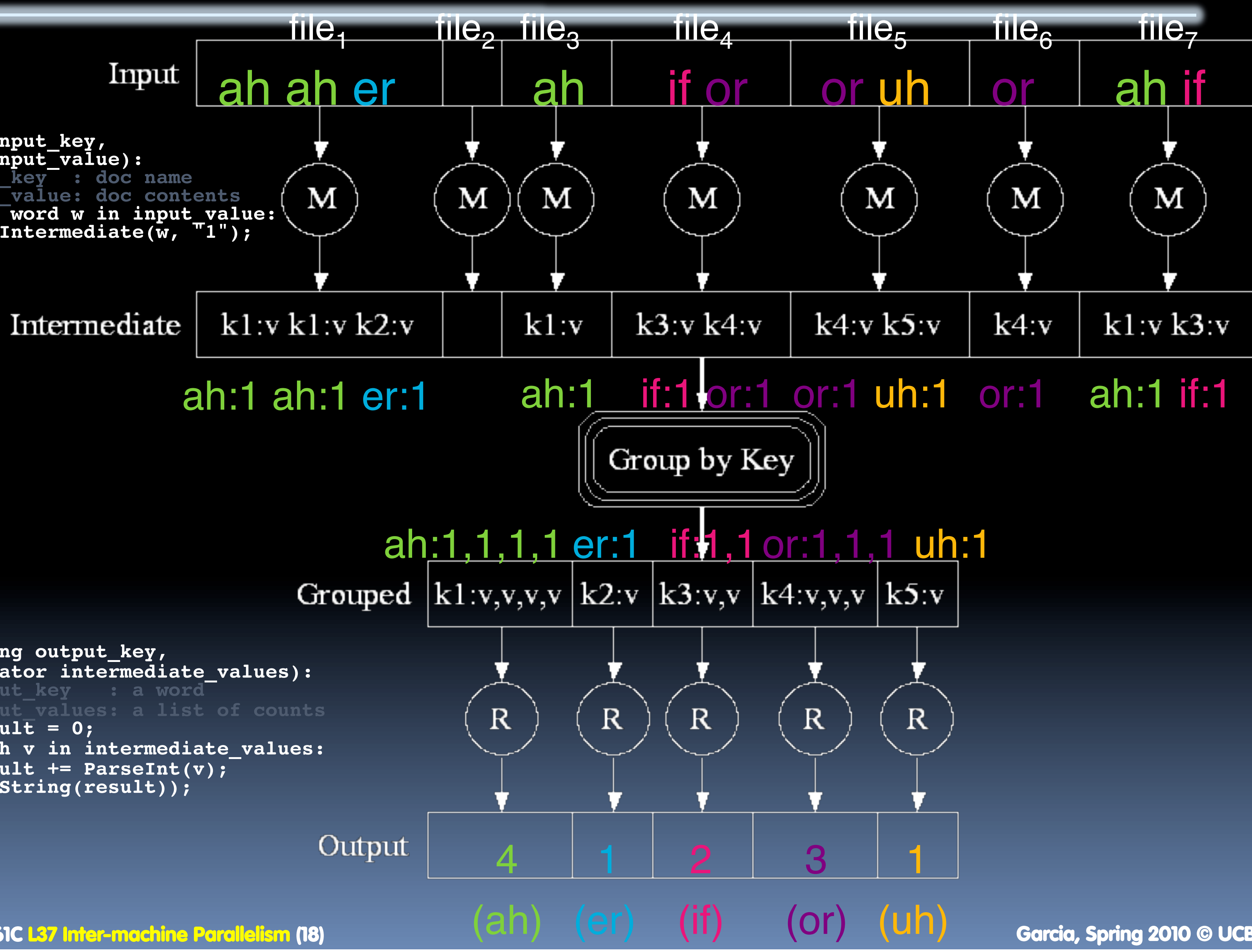
- "Reducer" nodes are responsible for the `reduce` function

```
// ("I",[1,1]) → ("I",2)
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));
```

- Data on a distributed file system (DFS)

MapReduce WordCount Diagram

```
map(String input_key,
    String input_value):
  // input_key  : doc name
  // input_value: doc 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));
```

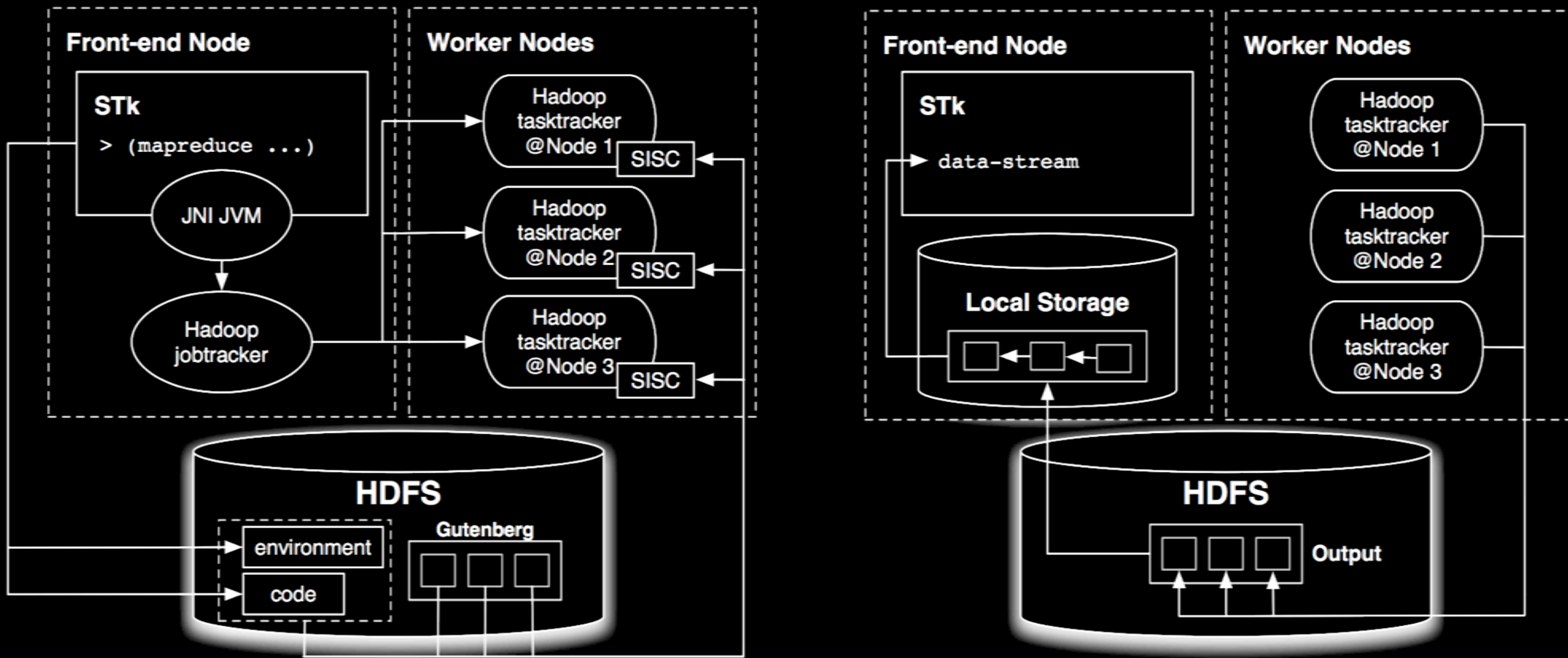
MapReduce WordCount Java code

```
public static void main(String[] args) throws IOException {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);
    conf.setMapperClass(WCMap.class);
    conf.setCombinerClass(WCReduce.class);
    conf.setReducerClass(WCReduce.class);
    conf.setInputPath(new Path(args[0]));
    conf.setOutputPath(new Path(args[1]));
    JobClient.runJob(conf);
}

public class WCMap extends MapReduceBase implements Mapper {
    private static final IntWritable ONE = new IntWritable(1);
    public void map(WritableComparable key, Writable value,
                    OutputCollector output,
                    Reporter reporter) throws IOException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            output.collect(new Text(itr.next()), ONE);
        }
    }
}

public class WCReduce extends MapReduceBase implements Reducer {
    public void reduce(WritableComparable key, Iterator values,
                       OutputCollector output,
                       Reporter reporter) throws IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += ((IntWritable) values.next()).get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
```

MapReduce in CS61A (and CS3?!)

- **Think that's too much code?**
  - So did we, and we wanted to teach the Map/Reduce programming paradigm in CS61A
    - "We" = Dan, Brian Harvey and ace undergrads Matt Johnson, Ramesh Sridharan, Robert Liao, Alex Rasmussen.
  - Google & Intel gave us the cluster you used in Lab!
- **You live in Scheme, and send the task to the cluster in the basement by invoking the fn `mapreduce`. Ans comes back as a stream.**
  - `(mapreduce mapper reducer reducer-base input)`
  - `www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-34.html`

Our Scheme MapReduce interface




```java
public static void main(String[] args) throws IOException {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);
    conf.setMapperClass(WCMap.class);
    conf.setCombinerClass(WCReduce.class);
    conf.setReducerClass(WCReduce.class);
    conf.setInputPath(new Path(args[0]));
    conf.setOutputPath(new Path(args[1]));
    JobClient.runJob(conf);
}

public class WCMap extends MapReduceBase implements Mapper {
    private static final IntWritable ONE = new IntWritable(1);
    public void map(WritableComparable key, Writable value,
                    OutputCollector output,
                    Reporter reporter) throws IOException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            output.collect(new Text(itr.next()), ONE);
        }
    }
}

public class WCReduce extends MapReduceBase implements Reducer {
    public void reduce(WritableComparable key, Iterator values,
                       OutputCollector output,
                       Reporter reporter) throws IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += ((IntWritable) values.next()).get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
```

```scheme
(define (mapper doc-line-pair)
  (map (lambda (wd) (make-kv-pair wd 1))
       (kv-value doc-line-pair)))

(mapreduce mapper + 0 "text-ID")
```

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, etc are handled
  - Can be much easier to design and program!
  - Can cascade several (many?) MapReduce tasks
- **But … it further restricts solvable problems**
  - Might be hard to express problem in MapReduce
  - Data parallelism is key
    - Need to be able to break up a problem by data chunks
  - MapReduce is closed-source (to Google) C++
    - Hadoop is open-source Java-based rewrite

Peer Instruction

1. Writing & managing **SETI@Home is relatively straightforward**; just hand out & gather data
2. The **majority of the world's computing power** lives in supercomputer centers

	12
a)	FF
b)	FT
c)	TF
d)	TT

Peer Instruction Answer

1. The heterogeneity of the machines, handling machines that fail, falsify data. FALSE
2. Have you considered how many PCs + game devices exist? Not even close. FALSE

1. Writing & managing SETI@Home is relatively straightforward; just hand out & gather data
2. The majority of the world's computing power lives in supercomputer centers

	12
a)	FF
b)	FT
c)	TF
d)	TT

Summary

- **Parallelism is necessary**
  - It looks like it's 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

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

To Learn More…

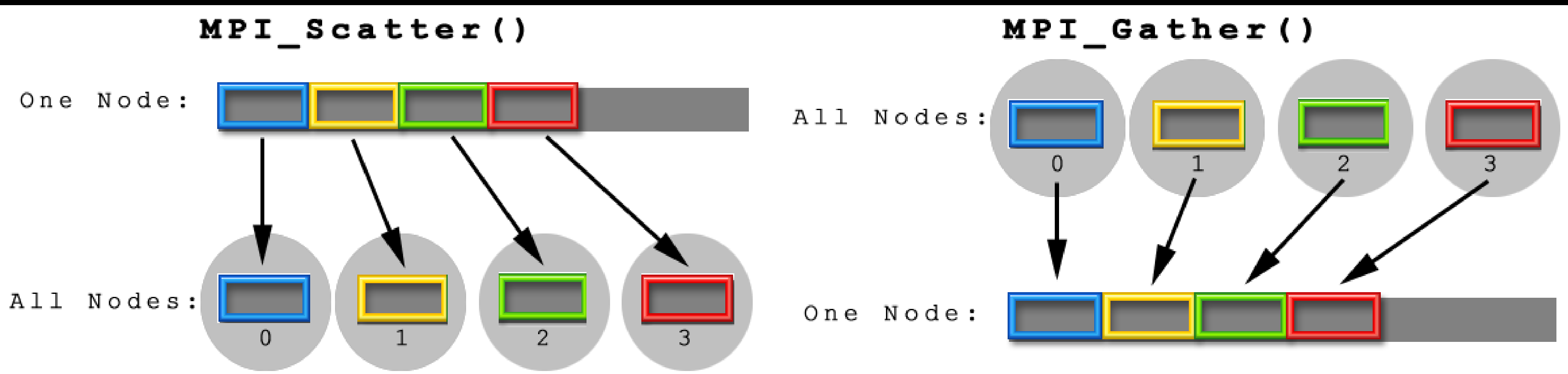
- **About MPI…**
  - `www.mpi-forum.org`
  - Parallel Programming in C with MPI and OpenMP by Michael J. Quinn
- **About MapReduce…**
  - `code.google.com/edu/parallel/mapreduce-tutorial.html`
  - `labs.google.com/papers/mapreduce.html`
  - `lucene.apache.org/hadoop/index.html`

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();
}
```