Disk Based Parallelism

1. Disk Based Parallelism
   - when random access memory is not large enough
   - an example: the pancake sorting problem
2. Roomy: A System for Space Limited Computations
   - extending C/C++ with a software library
   - programming with Roomy
3. Big Data
   - Hadoop and the Map/Reduce model

MCS 572 Lecture 23
Introduction to Supercomputing
Jan Verschelde, 14 October 2016

Disk Based Parallelism

processing large volumes of data

For the context of this lecture, consider:

- big data applications

  In applications with huge volumes of data,
  the bottleneck is not the number of arithmetical operations.

- the disk is the new RAM

  `Roomy` is a software library that allows to treat disk
  as Random Access Memory.

An application of `Roomy` concerns the determination
of the minimal number of moves to solve Rubik's cube.

Parallel Disk-based Computation

Parallel disk-based computation: instead of Random Access Memory, use disks as the main working memory of a computation.
This gives much more space for the same price.

Performance issues and solutions:

- Bandwidth: the bandwidth of a disk is roughly 50 times less than RAM (100 MB/s versus 5 GB/s).

  Solution: use many disks in parallel.

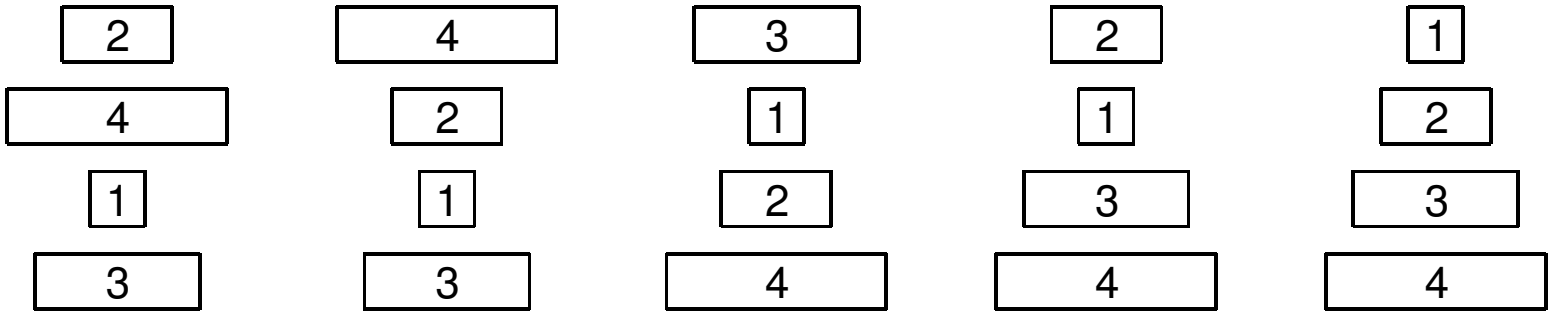
- Latency: even worse, the latency of disk is many orders of magnitude worse than RAM.

  Solution: avoid latency penalties by using streaming access.

Disk Based Parallelism

sorting pancakes

Given a stack of n numbered pancakes.
A spatula can reverse the order of the top k pancakes for $2 \leq k \leq n$.



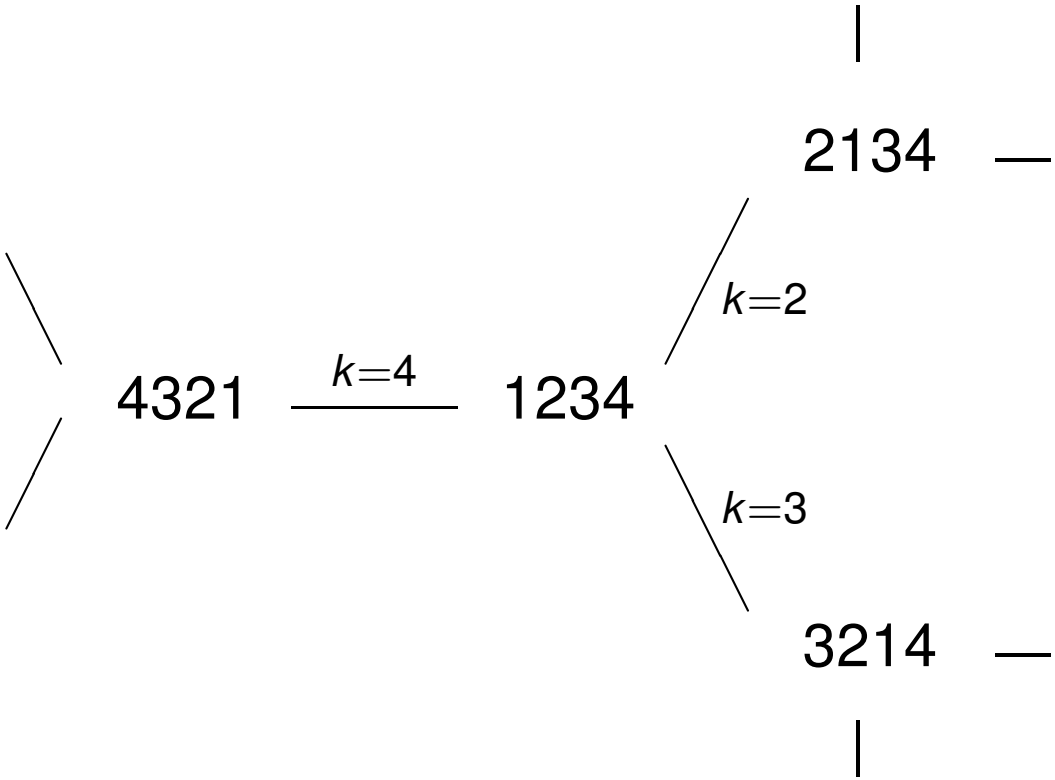
The sort happened with 4 flips:

1. k = 2
2. k = 4
3. k = 3
4. k = 2

How many flips (prefix reversals) are sufficient to sort?

pancake sorting graph

We generate all permutations using the flips:



There are 3 stacks that require 1 flip.

4! = 24 permutations, $24 = 1 + 3 + 6 + 11 + 3$,
at most 4 flips are needed for a stack of 4 pancakes.

breadth first search, running `pancake`

```
$ mpiexec -np 2 ./pancake
Sun Mar  4 13:24:44 2012: BEGINNING 11 PANCAKE BFS
Sun Mar  4 13:24:44 2012: Initial one-bit RoomyArray constructed
Sun Mar  4 13:24:44 2012: Level 0 done: 1 elements
Sun Mar  4 13:24:44 2012: Level 1 done: 10 elements
Sun Mar  4 13:24:44 2012: Level 2 done: 90 elements
Sun Mar  4 13:24:44 2012: Level 3 done: 809 elements
Sun Mar  4 13:24:44 2012: Level 4 done: 6429 elements
Sun Mar  4 13:24:44 2012: Level 5 done: 43891 elements
Sun Mar  4 13:24:45 2012: Level 6 done: 252737 elements
Sun Mar  4 13:24:49 2012: Level 7 done: 1174766 elements
Sun Mar  4 13:25:04 2012: Level 8 done: 4126515 elements
Sun Mar  4 13:25:45 2012: Level 9 done: 9981073 elements
Sun Mar  4 13:26:48 2012: Level 10 done: 14250471 elements
Sun Mar  4 13:27:37 2012: Level 11 done: 9123648 elements
Sun Mar  4 13:27:53 2012: Level 12 done: 956354 elements
Sun Mar  4 13:27:54 2012: Level 13 done: 6 elements
Sun Mar  4 13:27:54 2012: Level 14 done: 0 elements
Sun Mar  4 13:27:54 2012: ONE-BIT PANCAKE BFS DONE
Sun Mar  4 13:27:54 2012: Elapsed time: 3 m, 10 s, 536 ms, 720 us
```

how many flips to sort a stack of 11 pancakes?

$11 \times 10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 = 39,916,800$

#moves	#stacks
0	1
1	10
2	90
3	809
4	6429
5	43891
6	252737
7	1174766
8	4126515
9	9981073
10	14250471
11	9123648
12	956354
13	6
total	39916800

$\Rightarrow$ at most 13 moves for a stack of 11 pancakes.

Disk Based Parallelism

1. Disk Based Parallelism
   - when random access memory is not large enough
   - an example: the pancake sorting problem
2. Roomy: A System for Space Limited Computations
   - extending C/C++ with a software library
   - programming with Roomy
3. Big Data
   - Hadoop and the Map/Reduce model

Roomy

Roomy is

- a new programming model that extends a programming language with transparent disk-based computing supports;
- an open source C/C++ library implementing this new programming language extension;
- available at `sourceforge`;
- written by Daniel Kunkle in the lab of Gene Cooperman.

Applied to problems in computational group theory, and large enumerations, e.g.: Rubik's cube can be solved in 26 moves or less.

disk is the new RAM

Example: 50-node cluster with 200GB disk space per computer gives 10TB per computer. Bandwidth of one disk: 100MB/s, bandwidth of 50 disks in parallel: 5GB/s

$\Rightarrow$ The 10TB disk space is considered as RAM.

Some caveats:

- Disk latency remains limiting.
  Use "old-fashioned" RAM as cache.
  If disk is the new RAM, RAM is the new cache.
- Networks must restructured to emphasize local access over network access.

the Roomy programming model

The Roomy programming programming model

- provides basic data structures: arrays, lists, and hash tables;
- transparently distributes data structures across many disks and performs operations on that data in parallel;
- immediately process streaming access operators;
- delays processing random operators until they can be performed efficiently in batch.
  For example: collecting and sorting updates to an array.

Disk Based Parallelism

1. Disk Based Parallelism
   - when random access memory is not large enough
   - an example: the pancake sorting problem
2. Roomy: A System for Space Limited Computations
   - extending C/C++ with a software library
   - programming with Roomy
3. Big Data
   - Hadoop and the Map/Reduce model

programming construct: `map`

```
RoomyArray* ra;
RoomyList* rl;
// Function to map over ra.
void mapFunc ( uint64 i, void* val )
{
   RoomyListadd(rl,val);
}

int main ( int argc, char **argv )
{
   Roomy_init(&argc,&argv);
   ra = RoomyArray_makeBytes("array",sizeof(uint64),100);
   rl = RoomyList_make("list",sizeof(uint64));
   RoomyArray_map(ra,mapFunc); // execute map
   RoomyList_sync(rl); // synchronize list for delayed ads
   Roomy_finalize();
}
```

programming construct: `reduce`

Computing the sum of squares of the elements in a RoomyList:

```
RoomyList* rl;  // elements of type int

// add square of an element to sum
void mergeElt ( int* sum, int* element )
{
   *sum += (*e) * (*e);
}

// compute sum of two partial answers
void mergeResults ( int * sum1, int* sum2 )
{
    *sum1 += *sum2;
}
int sum = 0;
RoomyList_reduce(rl,&sum,sizeof(int),
                 mergeElt,mergeResults);
```

programming construct: predicate

Predicates count the number of elements in a data structure that satisfy a Boolean function.

```
RoomyList* rl;

// predicate: return 1 if element is greater than 42
uint8 predFunc ( int* val )
{
   return (*val > 42) ? 1 : 0;
}

RoomyList_attachPredicate(rl,predFunc);

uint64 gt42 = RoomyList predicateCount(rl,predFunc);
```

programming construct: permutation multiplication

For permutations X, Y, Z on N elements length N do Z = X*Y as
for i=0 to N-1: Z[i] = Y[X[i]].

```
RoomyArray *X, *Y, *Z;
// access X[i]
void accessX ( unit64 i, uint64* x_i) {
   RoomyArray_access(Y, *x_i,&i, accessY);
}
// access Y[X[i]]
void accessY ( uint64 x_i, uint64* y_x_i, uint64* i )
{
   RoomyArray_update(Z,*i,y_x_i,setZ);
}
// set Z[i] = Y[X[i]]
void setZ ( uint64, i, uint64* z_i, uint64* y_x_i, uint64* z_i_NEW )
{
   *z_i_NEW = *y_x_i;
}
RoomyArray_map(X,accessX); // access X[i]
RoomyArray_sync(Y);        // access Y[X[i]]
RoomyArray_sync(Z);        // set Z[i] = Y[X[i]]
```

programming construct: set operations

Convert list to set:

```
RoomyList *A; // can contain duplicates
RoomyList_removeDupes(A); // is a set
```

Union of two sets $A \cup B$:

```
RoomyList *A, *B;
RoomyList_addAll(A,B);
RoomyList_removeDupes(A);
```

Difference of two sets $A \setminus B$:

```
RoomyList *A, *B;
RoomyList_removeAll(A,B);
```

The intersection $A \cap B$ is implemented as $(A \cup B) \setminus (A \setminus B) \setminus (B \setminus A)$.

programming construct: breadth-first search

Initialize the search:

```
// Lists of all elements, current, and next level
RoomyList* all = RoomyList_make("allLev",eltSize);
RoomyList* cur = RoomyList_make("lev0",eltSize);
RoomyList* next = Roomy_list_Make("lev1",eltSize);

// Function to produce next level from current
void genNext ( T elt )
{
   // user defined code to compute neighbors ...
   for nbr in neighbors
      RoomyList_add(next,nbr);
}
// add start element
RoomyList_add(all,startElt);
RoomyList_add(cur,startElt);
```

perform the search

```
// generate levels until no new states are found
while(RoomyList_size(cur))
{  // generate next level from current
   RoomyList_map(cur,genNext);
   RoomyList_sync(next);
   // detect duplicates within next level
   RoomyList_removeDupes(next);
   // detect duplicates from previous levels
   RoomyList_removeAll(next,all);
   // record new elements
   RoomyList_addAll(all,next);
   // rotate levels
   RoomyList_destroy(cur);
   cur = next;
   newt = RoomyList_make(levName,eltSize);
}
```

Disk Based Parallelism

1. Disk Based Parallelism
   - when random access memory is not large enough
   - an example: the pancake sorting problem
2. Roomy: A System for Space Limited Computations
   - extending C/C++ with a software library
   - programming with Roomy
3. Big Data
   - Hadoop and the Map/Reduce model

Large Data Files

Process big data with parallel and distributed computing:

- MySQL scales well.
- overview of Hadoop:

Hadoop has become the de facto standard for processing big data.

Hadoop is used by Facebook to store photos, by LinkedIn to generate recommendations, and by Amazon to generate search indices.

Hadoop works by connecting many different computers,
hiding the complexity from the user: work with one giant computer.

Hadoop uses a model called Map/Reduce.

RHadoop by Antonio Piccolboni is a project for R and Hadoop,
hosted at `https://github.com`.

the Map/Reduce model

Goal: help with writing efficient parallel programs.

Common data processing tasks: filtering, merging, aggregating data and many (not all) machine learning algorithms fit into Map/Reduce.

Two steps:

Map: Tasks read in input data as a set of records, process the records, and send groups of similar records to reducers.
The mapper extracts a key from each input record.
Hadoop will then route all records with the same key to the same reducer.

Reduce: Tasks read in a set of related records, process the records, and write the results.
The reducer iterates through all results for the same key, processing the data and writing out the results.

Strength: the map and reduce steps run well in parallel.

example: predicting user behavior

Problem: how likely is a user to purchase an item from a website?
Suppose we have already computed (maybe using Map/Reduce)
a set of variables describing each user:

- most common locations,
- the number of pages viewed,
- the number of purchases made in the past.

Calculate forecast using random forests:

- Random forests work by calculating a set of regression trees and then averaging them together to create a single model.
- It can be time consuming to fit the random trees to the data, but each new tree can be calculated independently.
- One way to tackle this problem is to use a set of map tasks to generate random trees, and then send the models to a single reducer task to average the results and produce the model.

suggested reading

- Daniel Kunkle. **Roomy: A C/C++ library for parallel disk-based computation, 2010**. `http://roomy.sourceforge.net/`.
- Daniel Kunkle and Gene Cooperman.
  Harnessing parallel disks to solve Rubik's cube.
  Journal of Symbolic Computation 44:872-890, 2009.
- Daniel Kunkle and Gene Cooperman.
  Solving Rubik's Cube: Disk is the new RAM.
  Communications of the ACM 51(4):31-33, 2008.
- Garry Turkington. Hadoop Beginner's Guide.
  `www.it-ebooks.info`.

Summary + Exercises

With Roomy we can solve problems that would exhaust RAM.
Computational group theory often requires enumeration.

Exercises:

1. Watch the YouTube google tech talk of Gene Cooperman on the application of disk parallelism to Rubik's cube.
2. Read the paper of Daniel Kunkle and Gene Cooperman that was published in the Journal of Symbolic Computation.

The midterm exam of Friday can be in class or take home.
You can postpone the decision till 12:50PM on Friday 21 October.