A Library of Parallel Algorithms for Generating Combinatorial Objects

Amit Jain
Dept. of Math. and Computer Sc.
Boise State University
Boise, Idaho

Elizabeth Elzinga
Dept. of Computer Science
Grand Valley State University
Allendale, Michigan

Background

Sequential algorithms.

- *Combinatorial Algorithms* by Albert Nijenhuis and Herbert S. Wilf. (2nd ed., 1978)
- The packages *Combinatorica* for *Mathematica* and *Combinat* for *Maple*.
- *The Stony Brook Algorithm Repository*, by Steven S. Skiena
  Home page: `http://www.cs.sunysb.edu/~algorith`

Parallel algorithms have been published for generating some of the combinatorial objects in research papers but there are no available implementations.

Initial Project Goals

- all/random subsets of an n-set.
- all/random k-subsets of an n-subset.
- all/random permutations of n objects.
- all/random partitions of an n-subset.

Underlying Hardware Technology

- Beowulf cluster of 8 dual-Pentium PCs with two 100 Mbs Fast Ethernet networks. The cluster is isolated from the rest of the department via a switch. Check out the book *How to Build a Beowulf* that just came out from MIT press.
- Beowulf home page: `http://www.beowulf.org`

Underlying Software Technology

- PVM (Parallel Virtual Machine) software. This software is based on message passing between processes which can be on any heterogenous collection of machines. The software runs on almost all kinds of machines. Hence any program written with PVM can be used by almost any combination of hardware platform and operating systems.
- PVM home page: `http://www.epm.ornl.gov/pvm_home.html`

Project Accomplishments

- Generate all subsets of an n-set. Three different algorithms have been implemented. These are:
  - A *direct* algorithm.
  - A *lexicographic* algorithm.
  - A *binary reflected gray code* based algorithm.

  The sequential generation algorithms were taken from **Wilf**. The algorithms for ranking and unranking (needed for the parallel generation) were designed by us.

- Generate k-subsets of an n-set. Two different approaches were implemented.
  - A *lexicographic* algorithm.
  - A *binary reflected gray code* based algorithm.

  The sequential generation algorithms were taken from **Wilf**. The algorithms for ranking and unranking (needed for the parallel generation) were designed by us for the gray code based algorithm. The ranking and unranking algorithms for the lexicographic generation were based on **Akl**. We did discover an error in the published algorithm that was communicated to the author.

- Generate all permutations of $\{1, 2, \ldots, n\}$. Two different parallel algorithms were implemented.
  - A *lexicographic* algorithm.
  - A *binary reflected gray code* based algorithm.

  The sequential gray-code based algorithm was taken from **Wilf**. The algorithms for ranking and unranking (needed for the parallel generation) were designed by us. The ranking and unranking algorithms for the lexicographic generation were based on the algorithms published in **Akl**.

Wilf *Combinatorial Algorithms* by Albert Nijenhuis and Herbert S. Wilf. (Academic Press, 2nd ed., 1978)

Akl *The Design and Analysis of Parallel Algorithms* by S. G. Akl (Prentice Hall, 1989). Also the papers referred to in the book in Chapter 6.

Generation of all subsets of a set of given size

	size ($2^{size} subsets$)	sequential time (seconds)	parallel time (seconds)	speedup (with 16 CPUs)
direct:				
	22	1.14	0.20	5.7
	24	4.57	0.55	8.3
	26	18.25	1.93	9.5
	28	72.85	7.48	9.7
	30	291.26	30.11	9.7
	32	1165.89	113.79	10.2
lexicographic:				
	22	1.35	0.19	7.1
	24	4.77	0.53	9.0
	26	16.98	1.85	9.2
	28	67.90	7.16	9.5
	30	271.62	29.44	9.2
	32	1088.10	108.84	10.0
graycode:				
	22	1.02	0.19	5.4
	24	4.13	0.52	7.9
	26	16.57	1.92	8.6
	28	65.87	7.05	9.3
	30	265.72	28.08	9.5
	32	1057.77	107.26	9.9

Generation of all k-subsets of a set of size n

	size(n,k)	sequential time (seconds)	parallel time (seconds)	speedup (with 16 CPUs)
lexicographic:				
	24,12	1.05	0.23	4.6
	26,13	4.06	0.62	6.5
	28,14	15.68	2.14	7.3
	30,15	60.63	8.08	7.5
	32,16	242.23	31.00	7.8
	34,17	912.48	119.49	7.6
gray code:				
	24,12	0.72	0.17	4.2
	26,13	2.79	0.37	7.5
	28,14	10.78	1.19	9.1
	30,15	41.66	4.33	9.6
	32,16	161.67	16.75	9.7
	34,17	628.63	63.99	9.8

Experimental Setup

- The *sequential programs* were timed in the following environment:

  The compiler used was the GNU C++ compiler. The specific version used is specified below.

  ```
  g++ --version
  egcs-2.90.29 980515 (egcs-1.0.3 release)
  ```

  The programs were compiled and run under Red Hat Linux (version 5.2) running a 2.0.36 kernel (recompiled for SMP support).

  ```
  uname -a
  Linux kohinoor 2.0.36 #1 Sat Apr 10 17:46:43 MDT 1999 i586 unknown
  ```

  The machine used is a dual-processor with 166 MHz Pentiums with 64M of RAM.

- The *parallel programs* were implemented using PVM version 3.4.1 and timed on a 8-machine, 16-processor Beowulf cluster connected by two 100 Mbits/s Fast Ethernet. Each machine is identical to the machine on which the sequential timing was performed.

  ```
  kohinoor:for i in 0 1 2 3 4 5 6 7
  > do
  > echo "On machine pp0$i";rsh pp0$i pvm <<end
  > version
  > end
  > done
  On machine pp00
  pvm> 3.4.1
  On machine pp01
  pvm> 3.4.1
  On machine pp02
  pvm> 3.4.1
  On machine pp03
  pvm> 3.4.1
  On machine pp04
  pvm> 3.4.1
  On machine pp05
  pvm> 3.4.1
  On machine pp06
  pvm> 3.4.1
  On machine pp07
  pvm> 3.4.1
  ```

Further Directions

- The current implementations of ranking and unranking are limited due to the limited precision of integers in C++. We have investigated and found a suitable library for unlimited precision arithmetic that can be used to remove this restriction. The implementations for generating the subsets does not depend on the precision of integers.
- Add more families of combinatorial objects to the library.
- A surprising discovery was that the speedup was not linear as was expected. A further direction would be to investigate how to improve the speedup.