HPC in Combinatorics: Application of Work-Stealing

Florent Hivert

LRI / Université Paris Sud 11 / CNRS

Mars 2015

Outline

1. Background: Sage/OpenDreamKit/HPC in Combinatorics
2. Recursively enumerated sets (RESets)
3. The problem: Map/Reduce on RESets
4. A Python Implementation of Map/Reduce on RESets
5. HPC with Cilk/SIMD

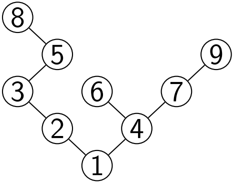
Algebraic combinatorics

Deep relations

$$\boxed{\text{behavior of algorithms}} \iff \boxed{\text{algebraic identities}}$$

Algebraic computation **using** / **on** / **combinatorial objects**

(()(()())(()))(()())



5				
2	6	9		
1	3	4	7	8

Algebraic combinatorics: experimental mathematics

A very large range of tools mathematical tools are needed:

- Manipulation of combinatorial objects:
  - integer partitions, set partitions, permutations, trees, ...
  - words, languages, automatons, ...
  - relations, graphs, partial orders, ...
- fast exact linear algebra over various rings
- commutative or not algebra (polynomials, series, ...)
- advanced algebraic computations (groups, group algebra, modules ...).

Together with very good language support for:

- advanced programming concept (objects, aspects, closures ...).
- basic persistent data structures, databases
- multicore, distributed computation

Algebraic combinatorics: experimental mathematics

A very large range of tools mathematical tools are needed:

- Manipulation of combinatorial objects:
  - integer partitions, set partitions, permutations, trees, ...
  - words, languages, automatons, ...
  - relations, graphs, partial orders, ...
- fast exact linear algebra over various rings
- commutative or not algebra (polynomials, series, ...)
- advanced algebraic computations (groups, group algebra, modules ...).

Together with very good language support for:

- advanced programming concept (objects, aspects, closures ...).
- basic persistent data structures, databases
- multicore, distributed computation

Algebraic combinatorics: experimental mathematics

We code primarily for research

- rapid prototyping
- 90% of the code is thrown away

- need high level, expressive language and libraries
- the code should be as close as possible to maths
- mathematical modeling

Combinatorial explosion

- We need the code to be reasonably efficient
- Everything which allows to speed-up high level code is good!

Algebraic combinatorics: experimental mathematics

We code primarily for research

- rapid prototyping
- 90% of the code is thrown away

- need high level, expressive language and libraries
- the code should be as close as possible to maths
- mathematical modeling

Combinatorial explosion

- We need the code to be reasonably efficient
- Everything which allows to speed-up high level code is good!

Algebraic combinatorics: experimental mathematics

We code primarily for research

- rapid prototyping
- 90% of the code is thrown away

- need high level, expressive language and libraries
- the code should be as close as possible to maths
- mathematical modeling

Combinatorial explosion

- We need the code to be reasonably efficient
- Everything which allows to speed-up high level code is good!

Today: Map/Reduce of RESets

Perform a map/reduce operation on a very large set described recursively.

- Typically the sets doesn't fit in the computer memory.

- Compute the cardinality
- Compute any kind of generating series
- Test a conjecture: i.e. find an element of S satisfying a specific property, or check that all of them do
- Count/list the elements of S having this property

Today: Map/Reduce of RESets

Perform a map/reduce operation on a very large set described recursively.

- Typically the sets doesn't fit in the computer memory.

- Compute the cardinality
- Compute any kind of generating series
- Test a conjecture: i.e. find an element of S satisfying a specific property, or check that all of them do
- Count/list the elements of S having this property

Today: Map/Reduce of RESets

Inputs:

A **recursively enumerated set**

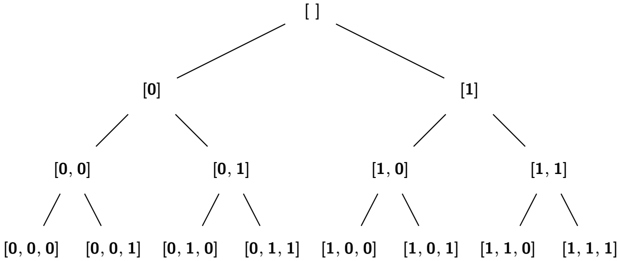
- the `roots` of the recursion
- the `children` function
- the `postprocessing` function

A **Map/Reduce problem**

- the `mapped` function
- the `reduce_init` function
- the `reduce` function

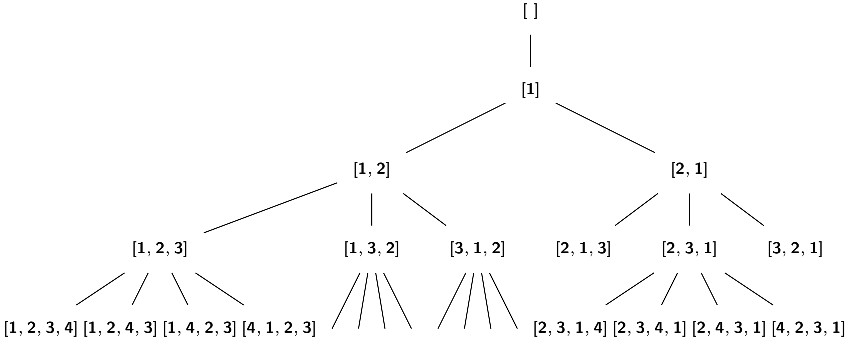
Examples of recursively enumerated sets (RESets)

Binary words: generation tree



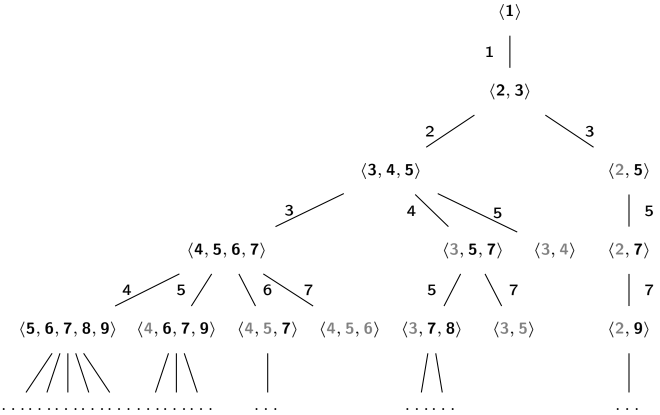
Examples of recursively enumerated sets (RESets)

Permutations: generation tree



Examples of recursively enumerated sets (RESets)

The tree of numerical semigroups



Map/Reduce on RESets

```
sage: S = RecursivelyEnumeratedSet(
....:   [[]],
....:   lambda l: [l+[0], l+[1]] if len(l) <= 15 else [],
....:   structure='forest', enumeration='depth')
sage: S.map_reduce(
....:   map_function = lambda x: 1,
....:   reduce_function = lambda x,y: x+y,
....:   reduce_init = 0 )
131071
```

Parallelism in Python

CPython has a Global interpreter lock (GIL)!

- No Parallel thread execution
  - Note: Python's GC uses reference counting, therefore the destructor `__del__` isn't thread-safe
  - Note: it is possible to release the GIL in C modules

Solution:

- multiprocess with several Python interpreters with IPC
- serialization (pickling in Python's dialect)
- Uses the `multiprocessing` module

Parallelism in Python

CPython has a Global interpreter lock (GIL)!

- No Parallel thread execution
  - Note: Python's GC uses reference counting, therefore the destructor `__del__` isn't thread-safe
  - Note: it is possible to release the GIL in C modules

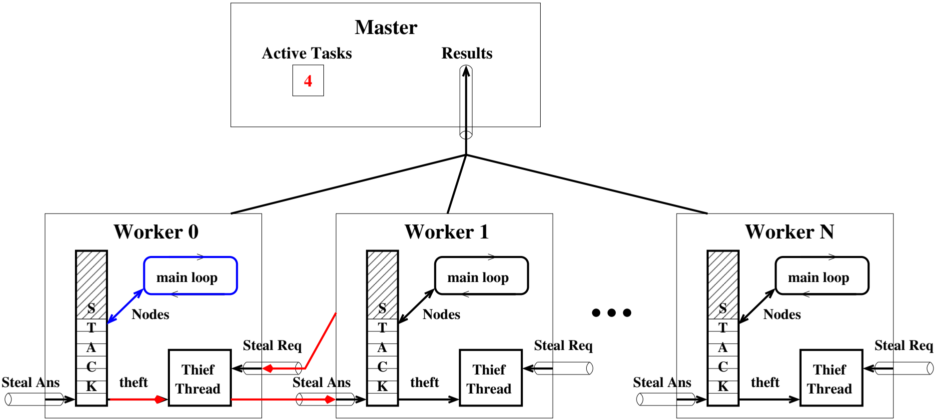
Solution:

- multiprocess with several Python interpreters with IPC
- serialization (pickling in Python's dialect)
- Uses the `multiprocessing` module

Implementation in Python using multiprocessing

- work stealing algorithm (Leiserson-Blumofe / Cilk)
- one process by worker
- communication by pipes and serialization
- thread used for thief

Work-Stealing System Architecture



When we really need speed!

Cython: optimising static compiler for both **Python** and extended Cython programming language.

- write Python code that calls back and forth from and to C or C++ code natively at any point.
- easily tune readable Python code into plain C performance by adding **static type declarations**.

Cilk: multithreaded parallel computing

- based on the C and C++ programming languages
- constructs to express parallel loops and the fork–join idiom.

Cilk extensions module for Python/Sage

When we really need speed!

Cython: optimising static compiler for both **Python** and extended Cython programming language.

- write Python code that calls back and forth from and to C or C++ code natively at any point.
- easily tune readable Python code into plain C performance by adding **static type declarations**.

Cilk: multithreaded parallel computing

- based on the C and C++ programming languages
- constructs to express parallel loops and the fork–join idiom.

Cilk extensions module for Python/Sage

When we really need speed!

Cython: optimising static compiler for both **Python** and extended Cython programming language.

- write Python code that calls back and forth from and to C or C++ code natively at any point.
- easily tune readable Python code into plain C performance by adding **static type declarations**.

Cilk: multithreaded parallel computing

- based on the C and C++ programming languages
- constructs to express parallel loops and the fork–join idiom.

Cilk extensions module for Python/Sage

Some results!

Computation of 16 days on a AMD Opteron(TM) Processor 6276, 2.3Gz using 32 cores. Generation of 80 Gbytes/s of combinatorial objects.

g	n_g	g	n_g	g	n_g
0	1	23	170 963	46	14 463 633 648
1	1	24	282 828	47	23 527 845 502
2	2	25	467 224	48	38 260 496 374
3	4	26	770 832	49	62 200 036 752
4	7	27	1 270 267	50	101 090 300 128
5	12	28	2 091 030	51	164 253 200 784
6	23	29	3 437 839	52	266 815 155 103
7	39	30	5 646 773	53	433 317 458 741
8	67	31	9 266 788	54	703 569 992 121
9	118	32	15 195 070	55	1 142 140 736 859
10	204	33	24 896 206	56	1 853 737 832 107
11	343	34	40 761 087	57	3 008 140 981 820
12	592	35	66 687 201	58	4 880 606 790 010
13	1 001	36	109 032 500	59	7 917 344 087 695
14	1 693	37	178 158 289	60	12 841 603 251 351
15	2 857	38	290 939 807	61	20 825 558 002 053
16	4 806	39	474 851 445	62	33 768 763 536 686
17	8 045	40	774 614 284	63	54 749 244 915 730
18	13 467	41	1 262 992 840	64	88 754 191 073 328
19	22 464	42	2 058 356 522	65	143 863 484 925 550
20	37 396	43	3 353 191 846	66	233 166 577 125 714
21	62 194	44	5 460 401 576	67	377 866 907 506 273
22	103 246	45	8 888 486 816		

The future

- Better integration Sage/Python/Cython/Cilk
- Generation graph (not tree!) in parallel?

- Trac Ticket 13580
  `http://trac.sagemath.org/ticket/13580`
- *Exploring the Tree of Numerical Semigroups* Jean Fromentin and Florent Hivert
  `https://hal.inria.fr/UNIV-ROUEN/hal-00823339v3`