Solving combinatorial problems on large multiGPU clusters: breaking the challenge of the Langford problem

Julien Loiseau
Christophe Jaillet
François Alin
Michaël Krajecki

CReSTIC - ROMEO
Université de Reims Champagne-Ardenne

Journée ROMEO, 11 juin 2015

Plan

combinatorial problem

- discrete problem
- combinatorial explosion
- combinatorial optimization / combinatorial search

- ? satisfiable
- build one or all the solutions
- determine the number of solutions

CSP = Constraint Satisfaction Problem

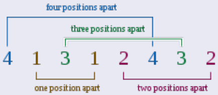
- combinatorial problem ⇒ NP-Complete
- NP-Complete ⇒ SAT/CSP
- tree representation

Langford problem

- $L(2,3) = 1$

  Yellow | Red | Blue | Red | Yellow | Blue

- numeric representation



- if and only if :
  $n = 4k$
  or $n = 4k - 1$
- $L(2,19)$ in 1999
  - sequential : 2 years $\frac{1}{2}$
  - distributed : 2 months
- $L(2,20)$ in 2002
  - algebraic method

number of solutions

n	Solutions
3	1
4	1
7	26
8	150
11	17792
12	108144
15	39809640
16	326721800
19	256814891280
20	2636337861200
23	3799455942515480
24	46845158056515900
27	??

Plan

parallelism

- nodes ⇔ interconnect
- machines
- 3 levels of parallelism

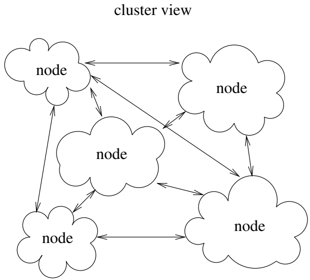
ROMEO

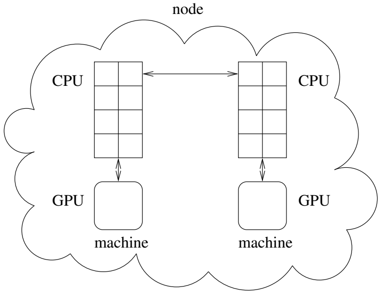
- Fat-tree with InfinyBand

  \- CPU : E5-2650v2 2.6GHz, 8c

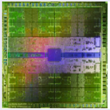
  \- GPU : NVIDIA K20Xm

→ TOP500 and GREEN500

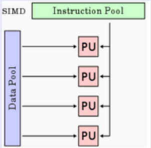




GPU



- SIMD/SIMT



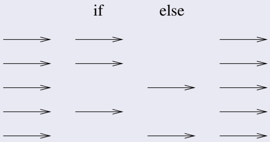
- 1000+ elementary processors
  - specific processors
  - simplified
  - synchronization

NVIDIA/CUDA

- hierarchical memory
- threads, blocks and grid
- warps : 32 threads

divergence

- SIMT, synchronization



- avoid desynchronization

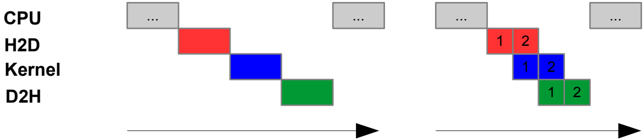
C Code

```
void saxpy(int n, float a, float *x,
           float *y)
{
      for(int i = 0 ; i < n ; i++)
      {
            y[i] = a*x[i] + y[i] ;
      }
}
...
int N = 1<<20 ;


saxpy(N, 2.3, x, y) ;
```

Cuda Code

```
__global__
void saxpy(int n, float a, float *x,
           float *y)
{
   int i = blockIdx.x*blockDim.x
           +threadIdx.x ;
   if(i < n) y[i] = a*x[i] + y[i] ;
}
...

int N = 1<<20 ;
cudaMemcpy(d_x, x, N,
           cudaMemcpyHostToDevice) ;
saxpy<<<4096,256>>>(N, 2.3, d_x, d_y) ;
cudaMemcpy(y, d_y, N,
           cudaMemcpyDeviceToHost) ;
```

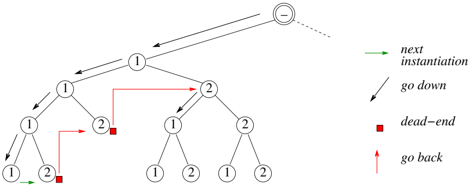
Streams

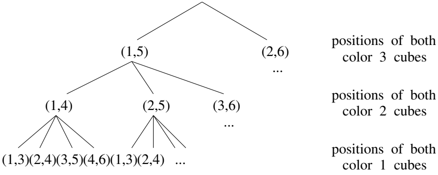


Plan

CSP resolution

- AC / ... / backtrack/backjumping/forwardchecking/...
- variable order, choice heuristic ...





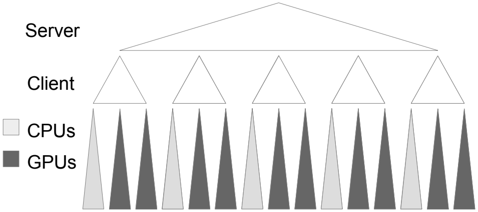
Langford problem $L(2,3)$

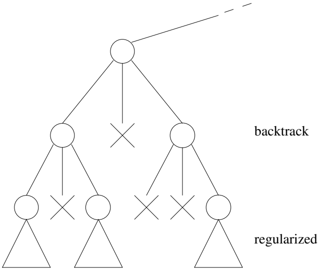
- level $\Rightarrow$ pair
- position conflict

parallel resolution

depth distribution

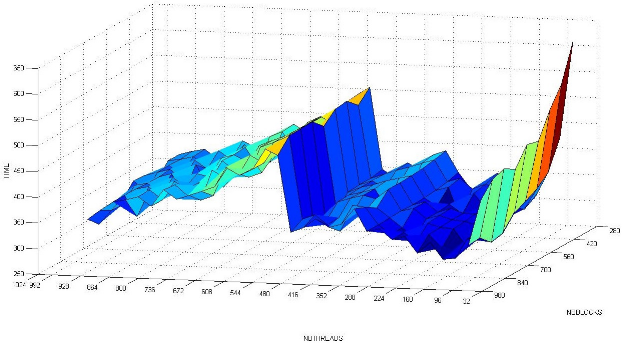
- p (server) + q (client) + r (CPU/GPU) = n
- large number of tasks ⇒ load balancing
- server, client and CPU ⇒ backtrack
- GPU : backtrack or vectorized

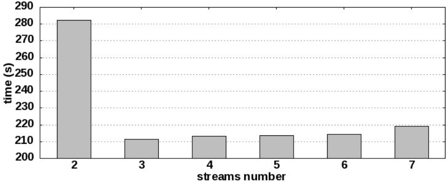


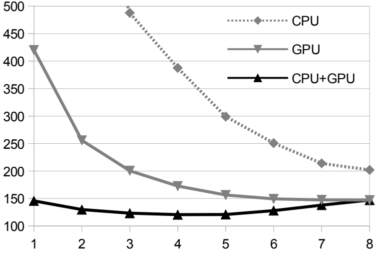


general methodology

- blocks and grid size ⇒ registers
- streams
- CPU cores involved to feed the GPU
- distribution depths







Factor	Backtrack	Regularized	Factor	Backtrack	Regularized
Threads per block	64-96	64-96	Server depth	3-4	3-4
Streams	1	3	CPU/GPU depth	9	5
CPU cores for GPU	$\emptyset$	3-4	Tasks distribution	80% for GPU	-

40 machines + 1 server :

n	Backtrack CPU	Regularized CPU + GPU	Backtrack CPU + GPU
16	21.219	14.344	6.637
17	200.306	120.544	37.166
18	1971.019	1178.261	408.501
19	22594.221	13960.871	4602.294

regularized

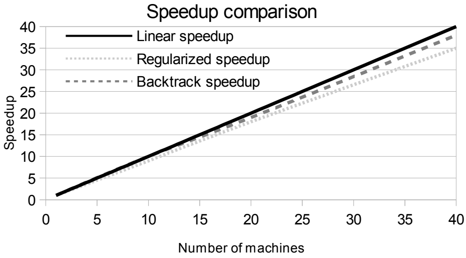
- $\approx$ CPU backtracking
- $\times$200 000 nodes
- GPU: 80% of the computation

backtrack GPU

- 3$\times$ faster
- GPU: 65% of the computation

258 machines + 1 server :

n	CPU	CPU + GPU
17	29.847	7.3
18	290.052	73.604
19	3197.526	803.524
20	–	9436.961
21	–	118512.420



258 nodes on ROMEO

- Miller's method previous limits : $L(2,19)$
- now $L(2,20)$ and $L(2,21)$
- 258 machines → speedup 230

Plan

algebraic method

- specific for the Langford problem
- based on cubes' positions
- simplifications

$$L(2,3) \Rightarrow X = (X_1, X_2, X_3, X_4, X_5, X_6)$$

$$F(X,3) = (X_1X_3 + X_2X_4 + X_3X_5 + X_4X_6) \times (X_1X_4 + X_2X_5 + X_3X_6)$$

$$\times (X_1X_5 + X_2X_6) = \prod_{i=1}^{n} \sum_{k=1}^{2n-i-1} x_k x_{k+i+1}$$

$$\sum_{(x_1,\ldots,x_{2n})\in\{-1,1\}^{2n}} \left(\prod_{i=1}^{2n} x_i\right) \prod_{i=1}^{n} \sum_{k=1}^{2n-i-1} x_k x_{k+i+1} = 2^{2n+1} L(2,n)$$

symmetry

- global sign changing $\Rightarrow F(-X, n) = F(X, n)$
- half sign changing $\Rightarrow$ pair or impair variables
- symmetry summing

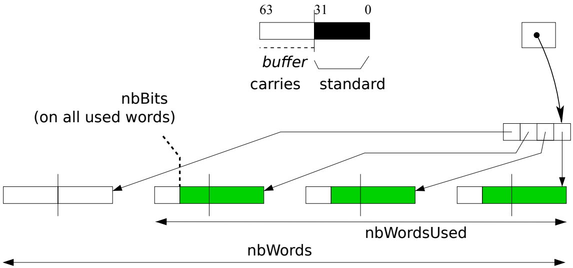
sum order

- change a single bit $\Rightarrow$ use the previous sum
- Gray code sequence

$$\begin{array}{c} 0\ 0\ 0\ 0 \Rightarrow 0 \\ 0\ 0\ 0\ 1 \Rightarrow 1 \\ 0\ 0\ 1\ 1 \Rightarrow 3 \\ 0\ 0\ 1\ 0 \Rightarrow 2 \\ \dots \end{array}$$

Big integer arithmetic

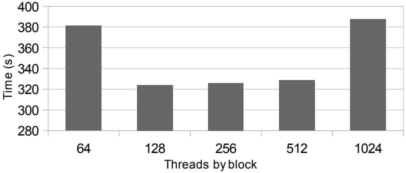
- $L(2, 16) \Rightarrow 70$ bits
- big integer representation needed
- specific for the problem

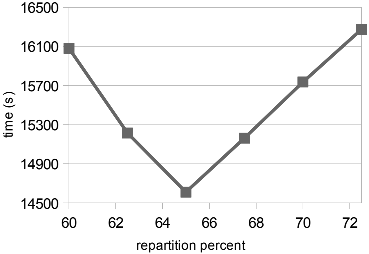


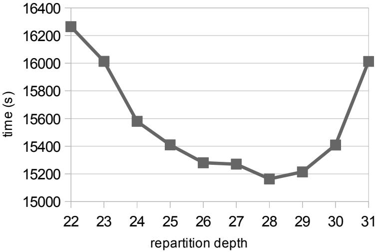
- in progress : assembly big integer on CPU/GPU

Experimental tuning

- → blocks and grid size
- → CPU/GPU distribution
- → distribution depth







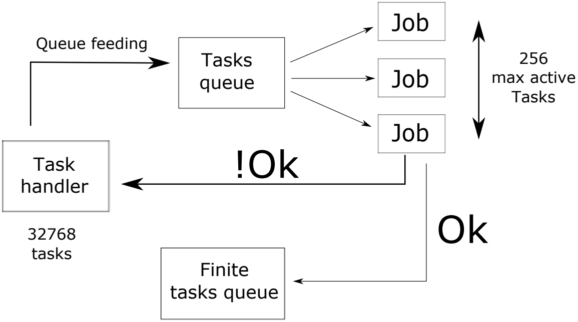
Workflow distribution

static distribution

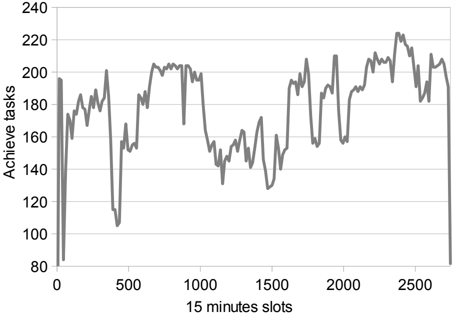
- MPI + [OpenMP/Cuda]
- one reservation

dynamic distribution

- Best-Effort + [OpenMP/Cuda]
- server + jobs
- requeue/cancel



$L(2,27)$ resolution using Best-Effort on ROMEO



→ 2 days of computation

→ 70% of ROMEO
$\approx$ 181 machines

$$L(2,27) = 111\ 683\ 606\ 778\ 027\ 803\ 456$$

Plan

backtrack resolution

- resolution methods using three levels of parallelism
- validation of the method
- Langford limit up to $L(2, 20)$-$L(2, 21)$

→ GPU efficiency: 80% of the computation

Godfrey's method

- Langford limit up to $L(2, 27)$
- GPU: 65% of the computation

perspectives

- solve $L(2, 28)$
- improve the method on other problems
- optimization problems