How to win Graph500
-- A Challenge to Graph500 Benchmark --

Katsuki Fujisawa

The Institute of Mathematics for Industry,
Kyushu University, Fukuoka, Japan

e-mail : fujisawa@imi.kyushu-u.ac.jp

October 2, 2015

Co@work, ZIB, Berlin
Graph500 benchmark

TUTORIAL
How to run the Graph500 benchmark

1. Installation

$ tar zxvf graph500_numa_opt_2013_coatwork.tar.gz
$ cd graph500_numa_opt_2013_coatwork
$ make

2. Execute the Graph500 benchmark

$ OMP_NUM_THREADS=2 ./graph500 -s 16

Usage

- Environment variables
  - OMP_NUM_THREADS=INT ... Number of threads
- Options
  - -s INT ... SCALE
  - -e INT ... Edgefactor (default 16)
Execution log

0. NETAL header

$ OMP_NUM_THREADS=4 ./graph500 -s 16

NETAL-BD13

Implementation: NETAL-BD13
ULIBC version: ULIBC (version 1.10, dummy)-HWLOC
RAM size: 0.00 GB
Number CPU procs: 4 (= 1 packages x 4 cores x 1 SMTs)
COMPILER: GCC 5.2.0

Graph instance: Kronecker2010
Scale, Edgefactor: 16 16
Energy-loop: disable
Number of threads: 4
Number of NUMA nodes: 1
Affinity(major:minor): disable:disable
Dir. parameters: 64 4
queue buffer size: 16384

This implementation is the updated version for following paper;
Execution log

1. Generating edge list

## -----------------------------------------------
## Kernel-0: Generating or Parsing edge list and Sampling source vertices
## -----------------------------------------------

[NUMA00 on Socket000] size(E): 1.05e+06, 0.01 GB (total: 0.01 GB)
Generating 'Kronecker2010' edges with (A,B,C,D) = (5700,1900,1900,500)
generating kronecker edge list ...

Ite. 01 of 10 generates 104858 edges (total: 104858) (0.019603 s, 5.349075e+06 E/s)
Ite. 02 of 10 generates 104858 edges (total: 209716) (0.016570 s, 6.328149e+06 E/s)
Ite. 03 of 10 generates 104858 edges (total: 314574) (0.019584 s, 5.354285e+06 E/s)
Ite. 04 of 10 generates 104858 edges (total: 419432) (0.015809 s, 6.632779e+06 E/s)
Ite. 05 of 10 generates 104858 edges (total: 524290) (0.016792 s, 6.244588e+06 E/s)
Ite. 06 of 10 generates 104858 edges (total: 629148) (0.015598 s, 6.722503e+06 E/s)
Ite. 07 of 10 generates 104857 edges (total: 734005) (0.017009 s, 6.164788e+06 E/s)
Ite. 08 of 10 generates 104857 edges (total: 838862) (0.018647 s, 5.623277e+06 E/s)
Ite. 09 of 10 generates 104857 edges (total: 943719) (0.014448 s, 7.257581e+06 E/s)
Ite. 10 of 10 generates 104857 edges (total: 1048576) (0.014475 s, 7.243953e+06 E/s)
done. (generated 1048576 edges)
Generated 'Kron2010' undirected 1048576 edges (SCALE 16.00, edgfactor 16.00: n=65536, m=1048576)
Takes 0.169004 seconds
## Kernel-1: graph construction

Constructing graph representation w/o self-loops and duplicated-edges...

Detecting graph size (0.007903 seconds)
- Number of subgraphs is 1
- Number of vertices is 65536
- Number of edges is 2096194
- Number of local edges is {E[0] has 65536 nodes and 2096194 edges,}
- Number of self-loops is 479
- Number of broken-edges is 0

Allocating NUMA graph representation...
- [NUMA00 on Socket000] G[0] 0.03 GB (total: 0.04 GB)
- done (0.000091 seconds)

G (N= 65536 nodes, M= 2096194 edges) = {
  G[0] = (FG: n=65536, m=2096194), (BG: n=65536 (off= 0), m=2096194 (n/N=100.0%, m/M=100.0%))
}

Constructing degree table...
- ell: 1
  - number_of_nodes: 65536
  - number_of_nonzero_nodes: 46802
  - number_of_zero_nodes: 18734
  - nonzero/total: 71.41418457 %
  - number_of_edges: 2096194
  - chunkszie: 65536

  max( degree[ 0 ] ): 25640

G[ 0 ] n: 65536, m: 2096194, nz(V): 46802, z(V): 18734

done (0.017838 seconds)
2. Constructing graph representation (continued)

....

Constructing CSR indices ... done. (0.000364 seconds)
Computing Prefix-sum for CSR index ... done. (0.003655 seconds)
Constructing NUMA-aware CSR graph ... done. (0.068494 seconds)
Sorting CSR values without duplication ...
    found and extracts 276312 (0.132 %) duplicated edges in Forward-graphs
    found and extracts 276312 (0.132 %) duplicated edges in Backward-graphs
done. (0.053454 seconds)
G = {
    G[0] = (FG: n=65536, m=1819882), (BG: n=65536 (off= 0), m=1819882 (n/N=100.0%, m/M=86.8%))
} = (N= 65536 nodes, M= 2096194 edges)
done.

Sorting edge list ...
Ite. 01 of 10 sorting 104858 edges (Elapsed: 0.008706 s, 0.008706 s, 1.204421e+07 E/s)
Ite. 02 of 10 sorting 209716 edges (Elapsed: 0.017373 s, 0.008660 s, 2.421708e+07 E/s)
Ite. 03 of 10 sorting 314574 edges (Elapsed: 0.026051 s, 0.008665 s, 3.630363e+07 E/s)
Ite. 04 of 10 sorting 419432 edges (Elapsed: 0.034673 s, 0.008612 s, 4.870232e+07 E/s)
Ite. 05 of 10 sorting 524290 edges (Elapsed: 0.044799 s, 0.010107 s, 5.187374e+07 E/s)
Ite. 06 of 10 sorting 629148 edges (Elapsed: 0.053712 s, 0.008876 s, 7.088315e+07 E/s)
Ite. 07 of 10 sorting 734005 edges (Elapsed: 0.062484 s, 0.008750 s, 8.388665e+07 E/s)
Ite. 08 of 10 sorting 838862 edges (Elapsed: 0.071179 s, 0.008686 s, 9.657825e+07 E/s)
Ite. 09 of 10 sorting 943719 edges (Elapsed: 0.079881 s, 0.008687 s, 1.086355e+08 E/s)
Ite. 10 of 10 sorting 1048576 edges (Elapsed: 0.088565 s, 0.008674 s, 1.208886e+08 E/s)
done. (0.088585 seconds)

Finished construction (0.313303 seconds).
MemoryUsage = 0.04 GB (47710208 bytes)
LocalMemoryUsages = {
    NUMA[0] 45.5 MB
}
3. Executing 64 BFSs

NUMA-optimized BFS (alpha=64, beta=4) ...

BFS( G(n=65536, m=2096194), s=1469 )

# i=01, s=1469, maxdist=5, time_s=0.002456, trav_e=1048569, teps=4.269091e+08

<table>
<thead>
<tr>
<th>Lv</th>
<th>algorithm</th>
<th>trav(ms)</th>
<th>swap(ms)</th>
<th>QF</th>
<th>Trav</th>
<th>Trav/QF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TopDown</td>
<td>0.078</td>
<td>0.116</td>
<td>1</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>1</td>
<td>BottomUp</td>
<td>0.834</td>
<td>0.026</td>
<td>25</td>
<td>528009</td>
<td>21120.4</td>
</tr>
<tr>
<td>2</td>
<td>BottomUp</td>
<td>0.402</td>
<td>0.031</td>
<td>8344</td>
<td>38855</td>
<td>4.65664</td>
</tr>
<tr>
<td>3</td>
<td>BottomUp</td>
<td>0.111</td>
<td>0.065</td>
<td>36611</td>
<td>1821</td>
<td>0.0497391</td>
</tr>
<tr>
<td>4</td>
<td>TopDown</td>
<td>0.135</td>
<td>0.000</td>
<td>1801</td>
<td>2055</td>
<td>1.14103</td>
</tr>
<tr>
<td>5</td>
<td>TopDown</td>
<td>0.159</td>
<td>0.000</td>
<td>6</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

$ Total 1.719 0.238 46788 570771$

trav: 1.719 ms, 70.0 %, swap: 0.238 ms, 9.7 %, init: 0.079 ms, 3.2 %, other: 0.421 ms, 17.1 %

# Local-neighbor-queue

<table>
<thead>
<tr>
<th>Lv</th>
<th>algorithm</th>
<th>QN[000]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TopDown</td>
<td>25</td>
</tr>
<tr>
<td>1</td>
<td>BottomUp</td>
<td>8344</td>
</tr>
<tr>
<td>2</td>
<td>BottomUp</td>
<td>36611</td>
</tr>
<tr>
<td>3</td>
<td>BottomUp</td>
<td>1801</td>
</tr>
<tr>
<td>4</td>
<td>TopDown</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>TopDown</td>
<td>0</td>
</tr>
</tbody>
</table>

# total 46787

# Local-traversal-edges

<table>
<thead>
<tr>
<th>Lv</th>
<th>algorithm</th>
<th>Trav[000]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TopDown</td>
<td>25</td>
</tr>
<tr>
<td>1</td>
<td>BottomUp</td>
<td>528009</td>
</tr>
<tr>
<td>2</td>
<td>BottomUp</td>
<td>38855</td>
</tr>
<tr>
<td>3</td>
<td>BottomUp</td>
<td>1821</td>
</tr>
<tr>
<td>4</td>
<td>TopDown</td>
<td>2055</td>
</tr>
<tr>
<td>5</td>
<td>TopDown</td>
<td>6</td>
</tr>
</tbody>
</table>

# total 570771

validating BFS tree ... (Tree:0.011s) ... (Tree2:0.000s) ... done (0.01 seconds)

| i=01, s=1469, maxdist=5, time_s=0.002456, trav_e=1048569, teps=4.269091e+08 |

- Search-direction
- CPU times and traverse and swap
- frontier-size and #traversed-edges on each NUMA node at each level

Number of next frontier nodes on each NUMA node at each level

Number of traversed edges on each NUMA node at each level

BFS-ID, Source vertex, Maximum level, CPU time of BFS in seconds, #Traversed-edges, TEPS

$TEPS = \frac{trav_e}{time_s}$
4. Graph500 results

**Parameters**
- SCALE and edgefactor (=16)
- Initial matrix (A,B,C,D) for Kronecker generator
- Number of sources (=64)

**Submit to Graph500 with**
- SCALE = 16
- edgefactor=16
- **Median TEPS = 374.3 MTEPS**

<table>
<thead>
<tr>
<th>SCALE:</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>edgefactor:</td>
<td>16</td>
</tr>
<tr>
<td>nvtx:</td>
<td>65536</td>
</tr>
<tr>
<td>terasize:</td>
<td>1.67772159999999997e-05</td>
</tr>
<tr>
<td>A:</td>
<td>5.7000000000000002e-02</td>
</tr>
<tr>
<td>B:</td>
<td>1.89999999999999995e-02</td>
</tr>
<tr>
<td>C:</td>
<td>1.89999999999999995e-02</td>
</tr>
<tr>
<td>D:</td>
<td>5.00000000000000010e-03</td>
</tr>
<tr>
<td>generation_time:</td>
<td>1.69003963470458984e-01</td>
</tr>
<tr>
<td>construction_time:</td>
<td>3.13302993774414062e-01</td>
</tr>
<tr>
<td>nbfs:</td>
<td>64</td>
</tr>
</tbody>
</table>

```
min_time: 0.0023191
firstquartile_time: 0.00245941
median_time: 0.00280106
thirdquartile_time: 0.00987935
max_time: 0.0562453
mean_time: 0.0100472
stddev_time: 0.0143069
min_nedge: 1048569
firstquartile_nedge: 1048569
median_nedge: 1048569
thirdquartile_nedge: 1048569
max_nedge: 1048569
mean_nedge: 1048569
stddev_nedge: 0
min_TEPS: 1.86428e+07
firstquartile_TEPS: 1.06137e+08
median_TEPS: 3.74347e+08
thirdquartile_TEPS: 4.2635e+08
max_TEPS: 4.52145e+08
harmonic_mean_TEPS: 1.04364e+08
harmonic_stddev_TEPS: 1.87233e+07
min_validate: 0.0108559
firstquartile_validate: 0.0109969
median_validate: 0.0111861
thirdquartile_validate: 0.0133055
max_validate: 0.0163701
mean_validate: 0.0121689
stddev_validate: 0.00148899
```
Performance comparison

• Median GTEPS (=10⁹ TEPS) on a NUMA-based server
  – GCC 4.4.7
  – 4-way Intel Xeon E5-4640 2.40GHz and 512 GB RAM

<table>
<thead>
<tr>
<th>Scale</th>
<th>Reference code</th>
<th>Graph500-BD2013</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0.358</td>
<td>4.578</td>
<td>12.8x</td>
</tr>
<tr>
<td>21</td>
<td>0.233</td>
<td>6.317</td>
<td>27.1x</td>
</tr>
<tr>
<td>22</td>
<td>0.101</td>
<td>8.433</td>
<td>83.7x</td>
</tr>
<tr>
<td>23</td>
<td>0.129</td>
<td>10.023</td>
<td>77.6x</td>
</tr>
<tr>
<td>24</td>
<td>0.123</td>
<td>10.829</td>
<td>88.3x</td>
</tr>
<tr>
<td>25</td>
<td>0.107</td>
<td>11.155</td>
<td>104.4x</td>
</tr>
<tr>
<td>26</td>
<td>0.097</td>
<td>11.149</td>
<td>114.9x</td>
</tr>
<tr>
<td>27</td>
<td>0.087</td>
<td>10.854</td>
<td>124.5x</td>
</tr>
<tr>
<td>28</td>
<td>–</td>
<td>9.887</td>
<td>–</td>
</tr>
<tr>
<td>29</td>
<td>–</td>
<td>9.393</td>
<td>–</td>
</tr>
</tbody>
</table>

Segmentation fault

100+ faster!
NETAL for Graph Analysis

TUTORIAL
Tutorial of NETAL (1)

0. Build on Linux and OSX

$ tar zxvf netal-1.51-coatwork.tar.gz && cd netal-1.51-coatwork && make

※ We strongly recommend to use Homebrew-GCC or Macports-GCC on OSX

1. Computes Betweenness centrality

$ ./netal -i data/sample.gr -t dimacs -z cent uB 1.00 -o out_uB.sp
Tutorial of NETAL (1)

0. Build on Linux and OSX

$ tar zxvf netal-1.51-coatwork.tar.gz && cd netal-1.51-coatwork && make
※ We strongly recommend to use Homebrew-GCC or Macports-GCC on OSX

1. Computes Betweenness centrality

$ ./netal -i data/sample.gr -t dimacs -z cent uB 1.00 -o out_uB.sp

sample.gr (DIMACS format graph)

This graph contains 9 nodes and 12 edges

Each line describes a directed weighted edge; (Tail-ID, Head-ID, Weight)

※ Index starts from 0

http://www.dis.uniroma1.it/challenge9/format.shtml
Tutorial of NETAL (1)

0. Build on Linux and OSX

```bash
$ tar zxvf netal-1.51-coatwork.tar.gz && cd netal-1.51-coatwork && make
```

※ We strongly recommend to use Homebrew-GCC or Macports-GCC on OSX

1. Computes Betweenness centrality

```bash
$ ./netal -i data/sample.gr -t dimacs -z cent uB 1.00 -o out_uB.sp
```

sample.gr (DIMACS format graph)

```
p sp 9 12
a 1 2 2
a 1 3 5
a 1 4 3
a 2 5 8
a 3 7 1
a 4 7 2
a 4 9 6
a 5 3 4
a 5 6 6
a 6 8 7
a 7 8 9
a 9 8 2
```

This graph contains 9 nodes and 12 edges

Each line describes a directed weighted edge; (Tail-ID, Head-ID, Weight)

※ Index starts from 0

http://www.dis.uniroma1.it/challenge9/format.shtml

Accuracy 1.00 ... exact

u ... unweighted
B ... Betweenness centrality
Tutorial of NETAL (1)

0. Build on Linux and OSX

```
$ tar zxvf netal-1.51-coatwork.tar.gz && cd netal-1.51-coatwork && make
```

※ We strongly recommend to use Homebrew-GCC or Macports-GCC on OSX

1. Computes Betweenness centrality

```
$ ./netal -i data/sample.gr -t dimacs -z cent uB 1.00 -o out_uB.sp
```

sample.gr (DIMACS format graph)

```
p sp 9 12
a 1 2 2
a 1 3 5
a 1 4 3
a 2 5 8
a 3 7 1
a 4 7 2
a 4 9 6
a 5 3 4
a 5 6 6
a 6 8 7
a 7 8 9
a 9 8 2
```

※ Index starts from 0

http://www.dis.uniroma1.it/challenge9/format.shtml

out_uB.sp (NETAL format)

```
p sp 9 12
l Betweenness
L edge Betweenness
c
d 1 0
d 2 2
d 3 2.83333
d 4 2.16667
...
ed 1 2 3
ed 1 3 1.83333
ed 1 4 3.16667
ed 2 5 7
ed 3 7 4.83333
```

※ This graph contains 9 nodes and 12 edges

※ Each line describes a directed weighted edge;
   (Tail-ID, Head-ID, Weight)

※ This graph contains 9 nodes and 12 edges

(Node-ID, BC-score)

(Tail-ID, Head-ID, Edge-BC-score)
# Tutorial of NETAL (2)

## Betweenness centrality (Exact)

```bash
$ ./netal -i data/sample.gr -t dimacs -z cent uB 1.00 -o out_uB.sp
```

## Betweenness centrality (10%-approx. using random sampling of source vertices)

```bash
$ ./netal -i data/sample.gr -t dimacs -z cent uB 0.10 -o out_uB.sp
```

## Multiple centrality

Specify number of threads (default: #threads=logical cores on a system)

```bash
$ OMP_NUM_THREADS=8 ./netal -i data/sample.gr -z cent wmCGSBD 1.00 -o out.sp
```

<table>
<thead>
<tr>
<th>C</th>
<th>Closeness [1]</th>
<th>( C_C(v) = \frac{1}{\sum_{t \in V} d_G(v, t)} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Graph [2]</td>
<td>( C_G(v) = \frac{1}{\max_{t \in V} d_G(v, t)} )</td>
</tr>
<tr>
<td>S</td>
<td>Stress [3]</td>
<td>( C_S(v) = \sum_{s \neq v \neq t \in V} \sigma_{st}(v) )</td>
</tr>
<tr>
<td>B</td>
<td>Betweenness [4]</td>
<td>( C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} )</td>
</tr>
<tr>
<td>D</td>
<td>Degree [5]</td>
<td>( C_D(v) = \sum_{v \in V} \deg_G(v) )</td>
</tr>
</tbody>
</table>

### Options

- **m** ··· computes maximum connected components\(^*1\) only
- **u** ··· computes unweighted centrality
- **w** ··· computes weighted centrality

\(^*1\) Using Shiloach-Vishkin algorithm
$ OMP_NUM_THREADS=8 ./netal -i data/sample.gr -z cent wmCGSBD 1.00 -o out.sp

c file name: out.sp
instance name: sample.gr
created: Mon Feb 9 23:51:36 2015

p sp 9 12

c centrality (mwBCDGS, acc=100.000000%, weighted, random sampling) (|Vs|=9, exact)
diameter is 21 (s=1)

<table>
<thead>
<tr>
<th>Centrality scores for each node</th>
</tr>
</thead>
<tbody>
<tr>
<td>d 1 1.18033 3 0.5 0 0</td>
</tr>
<tr>
<td>d 2 1.05882 1 0.380952 3 3</td>
</tr>
<tr>
<td>d 3 6.54545 1 0.8 6 6</td>
</tr>
<tr>
<td>d 4 4.5 2 1 2 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Centrality scores for each edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>e 1 2 0 0</td>
</tr>
<tr>
<td>e 1 3 0 0</td>
</tr>
<tr>
<td>e 1 4 3 3</td>
</tr>
<tr>
<td>e 2 5 6 6</td>
</tr>
<tr>
<td>e 3 7 8 8</td>
</tr>
<tr>
<td>e 4 7 0 0</td>
</tr>
<tr>
<td>e 4 9 4 4</td>
</tr>
</tbody>
</table>

Centrality scores for each node

Centrality scores for each edge
Examples: Digraph with 9 nodes and 12 edges

Input digraph

Closeness

Graph

Degree

Stress

Betweenness
Degree centrality

Input digraph
• 9 nodes and 12 edges

Degree centrality
• Number of outgoing edges
**Closeness centrality**

**Input digraph**
- 9 nodes and 12 edges

**Closeness centrality**

\[ C_C(v) = \frac{n(n - 1)}{\sum_{t \in V} d_G(v, t)} \]
Betweenness centrality

Input digraph
• 9 nodes and 12 edges

Betweenness centrality

\[ C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} \]

Node: Betweenness (min: 0.00, max: 5.00)
Edge: edge-Betweenness (min: 1.83, max: 7.00)

Importance
Appendix for ULIBC

TUTORIAL
ULIBC on Virtual- and Real-Machine

• ULIBC
  – Detects processor topology at runtime
  – Constructs CPU and memory affinities automatically
  – Available at https://bitbucket.org/yuichiro_yasui/ulIBC

• How to build
  – on Virtual Machine (CO@Work)
    $ make CC=gcc-5 ULIBCDIR=./ULIBC_v1.10_dummy/
  – on Real Machine
    $ make CC=gcc-5 ULIBCDIR=./ULIBC_v1.10/