Parallel asynchronous algorithm for the search of connected components on multiprocessors

S.O. Komarov, NSU
K.V. Kalgin, PhD, ICMMG, SB RAS
Connected component

- In graph $G$, two vertices $u$ and $v$ are called **connected** if $G$ contains a path from $u$ to $v$.

- A graph is **connected** when any two vertices are connected.

- A connected component of a graph is a subgraph which any two vertices are connected, and which is connected to no additional vertices in the supergraph.
Graphs

Types of graphs:

- rmat-n \((2^n \text{ vertices})\)
- ssca2-n \((2^n \text{ vertices})\)
- grids + single vertexs
- linear

Graphs are given in CSR format
Testing on:
- *Intel Xeon E5-2690 2.90GHz 2х8 ядер*
- *Intel Xeon Phi 7110X*

Compilation keys in icpc:
- *O3 -ipo -march=native*

Core affinity:
- *Export KMP_AFFINITY=compact*
Algorithm Shiloach-Vishkin(SV)

Array D[1..n], where D[i] is the component’s number to which “i” is belong. Initially D[i] = i.

1) for (i, j) ∈ E do
   if( (D[i] < D[j]) && (D[j] == D[D[j]]) ) then D[D[j]] = D[i];

2) Pointer Jumper
   for (i, j) ∈ E do
      while( D[i] != D[D[i]] ) do D[i] = D[D[i]];

3) if( D not change) then exit;
   Otherwise, go to 1.
Algorithm Shiloach-Vishkin (SV)

+ Scalability

+ Easy to be parallelized
  a. insert openMP pragmas
  b. no dependencies

- Often repeats operations
Algorithm Shiloach-Vishkin (SV)

Xeon E5-2690

Phi 7110X

MTEPS vs THREADS for Xeon E5-2690 and Phi 7110X

- rmat-22
- scca2-22
- grid
Asynchronous algorithm (ABFS)

Dynamic pool of colors $P[1..n]$

1) Each thread executes BFS to paint some subgraph in its own color

2) Pointer Jumper

3) Coloring (simple loop)
ABFS(naive)+SV

- ABFS colors large components
  - if thread visits colored vertex, it skips

- SV makes corrections (at the border)
ABFS (naive) + SV

Xeon E5-2690

Phi 7110X

MTEPS

THREADS

THREADS

rama-22
sca2-22
grid
ABFS+SV

- ABFS colors large components
  - if thread visits colored vertex and vertex’s color less than thread’s color, then thread change own color

- SV makes corrections (at the border)
Each thread executes BFS to paint some subgraph
  o if thread visits colored vertex and vertex’s color less than thread’s color, then thread change own color
  o collisions are handled in critical sections

Pointer Jumper

Coloring
PBFS+lock free

- PBFS+critical
- Critical sections are replaced on built-in functions:
  
  ```
  bool __sync_bool_compare_and_swap(...)
  ```

  ```
  do {
    while( old_root != pool[ old_root ] ) old_root = pool[ old_root ];
    while( new_root != pool[ new_root ] ) new_root = pool[ new_root ];
    if( new_root < old_root )
      is_ok = __sync_bool_compare_and_swap( & pool[ old_root ], old_root, new_root );
    else
      is_ok = __sync_bool_compare_and_swap( & pool[ new_root ], new_root, old_root );
  } while( ! is_ok )
  ```
PBFS+lock free

*due to the low performance of this algorithm will not be considered further*
ABFS+SV - Hybrid

- ABFS colors large components
  - if thread visits colored vertex and vertex’s color less than thread’s color, then thread change own color
- A first phase of SV makes corrections (at the border)
- Pointer Jumper
- Coloring
ABFS+SV - Hybrid

Xeon E5-2690

MTEPS vs. THREADS

Phi 7110X

MTEPS vs. THREADS
Comparison of algorithms on rmat-22
Comparison of algorithms on ssc2a-22

**Xeon E5-2690**

**Phi 7110X**

- Hybrid
- ABFS+SV
- SV
- Naive
Comparison of algorithms on grid

**Xeon E5-2690**

**Phi 7110X**
Perspective

- Optimization under Xeon Phi
- Further improvements of the algorithm
  - early stop at the first stage of the ABFS
  - using block of colors by thread