OPTIMIZATION OF COMMUNICATION AND MEMORY ACCESS IN BFS BENCHMARK

March 5, 2015

Alexander Daryin
Head of Advanced Research Lab
alexander.daryin@t-platforms.ru
Introduction

- Breadth-First Search (BFS) on distributed memory systems
- Part of Graph500 benchmark

- Performance is limited by
  - memory access latency
  - network latency
  - network bisection bandwidth
Compressed Sparse Row Graph (CSR)

**Column**: 13 21 02 10 10 11 33 22 23 02 03 31

**Rowstarts**: 0 1 2 3

**Global vertex id**: 02 03 10 11 13 21 22 23

**Rank**

**Local id**: 30 31 32 33

64 bits

GraphHPC-2015
Reference Algorithm

Data: CSR (row starts R, column C), current queue Q (array)

1 function ProcessQueue(R, C, Q)

2      for v ∈ Q do

3          for e ← R[v] to R[v + 1] do

4              send v, local(C[e]) to owner(C[e]))
Reference Algorithm

Data: CSR (row starts R, column C), current queue Q (array)

1 function ProcessQueue(R, C, Q)
2     for v ∈ Q do
3         for e ← R[v] to R[v + 1] do
4             send v, local(C[e]) to owner(C[e])

Random memory access pattern
cache miss
**Reference Algorithm**

**Data:** CSR (row starts \( R \), column \( C \)), current queue \( Q \) (array)

```plaintext
1 function ProcessQueue(\( R, C, Q \))
2     for \( v \in Q \) do
3         for \( e \leftarrow R[v] \) to \( R[v+1] \) do
4             send \( v, \text{local}(C[e]) \) to owner(\( C[e] \))
```

- Small message size
- Low bandwidth
- Irregular communication pattern
- Connection overhead
- Message coalescing
- Separate send buffer for each peer process
- Large memory overhead
- Multiple threads - ?
Goals

- Optimize memory access
- Regularize communication pattern
- Reduce memory consumption
- Allow for multithreaded design

Possible solution:
- partition graph data for each peer node
- process partitions independently (and possibly concurrently)
Naïve Partitioning

Infeasible: $|\text{rowstarts}| = |V|$
Packed Partitioned CSR

\[ |\text{rowstarts}| \leq |\text{local edges}| + \text{NP} \]
Algorithm for Packed CSR

Data: Packed CSR (vertex indices $V$, row offsets $D$, column $C$), current queue $Q$ (bit mask), number of processes $NP$

1 function $\text{ProcessQueue}()$
2     for peer ← 0 to $NP$ do
3     for $i ← 0, |V[\text{peer}]|$ do
4         if $V[\text{peer}][i] \in Q$ then
5             for $e ← D[\text{peer}][i]$ to $D[\text{peer}][i + 1]$ do
6                 send $V[\text{peer}][i], C[e]$ to peer
Compressed Packed CSR

peer 0

2 3

peer 1

0 3 0 1

peer 2

0 1 2

peer 3

1 3 2 2

80% entries: 1 edge

compressed rowstarts

flag
local id | offset
immediate value flag

column

flag
local id
stop flag
Peer Visited Mask

- Do not repeat target vertices (in current round)
- 2x speedup
- Impossible with original non-partitioned CSR

```
0 3 0 1
```

compressed packed CSR

```
0 1
```

input queue mask

```
0 1 2 3
```

peer visited mask

```
0 0 0 3 1 0 1 1
```

messages sent to rank 1

```
0 0 0 3 1 0 1 1
```

messages sent to rank 1 using peer visited mask
Algorithm with Peer Visited Mask

Data: Packed CSR (vertex indices V, row offsets D, column C), current queue Q (bit mask), number of processes NP

```
1 function ProcessQueue()
2     for peer ← 0 to NP do
3         clear(P)            /* clear peer visited mask */
4         for i ← 0, |V[peer]| do
5             if V[peer][i] ∈ Q then
6                 for e ← D[peer][i] to D[peer][i + 1] do
7                     if C[e] ∉ P then
8                         set P[C[e]]
9                         send V[peer][i], C[e] to peer
```
Message Compression

- Do not repeat source vertices
- **Up to 40% traffic reduction** (in forward phase)

![Compressed packed CSR](image)

```
0 3 0 1
```

```
0 1
```

- `compressed packed CSR`

![Input queue mask](image)

```
0 1 2 3
```

- `input queue mask`

![Peer visited mask](image)

```
0 1 2 3
```

- `peer visited mask`

![Messages sent to rank 1](image)

```
0 0 0 3 1 1
```

- `messages sent to rank 1 without compression`

![Messages sent to rank 1 with compression](image)

```
0 0 3 1 1
```

- `messages sent to rank 1 with compression`

![Message format](image)

```
src tgt1 ... flag tgtK
```

- `message format`

- `stop flag`
Benefits of Packed CSR

- Sequential memory access
- Regular communication pattern
  - Fixed number of large send/receive buffers
  - Lower overhead incurred by network adapters
  - Use dynamically connected transport (DC/XRC) on modern InfiniBand hardware
- Reduced network traffic
- Individual CSRs may be processed concurrently
Additional Optimizations

- **Direction Optimization** (backward stepping)
- Blocks of packed CSRs:
  - *Coarse-grained queue mask* (forward phase)
  - *Coarse-grained visited mask* (backward phase)
- **Load balancing** (heavy vertices)
**Backward Stepping**

**Data:** Packed CSR (vertex indices $V$, row offsets $D$, column $C$), bit mask of visited vertices $M$, number of processes $NP$

```plaintext
1 function ProcessUnvisited()
   // probe first edges
   for peer ← 0 to $NP$ do
      for $i ← 0, |V[peer]|$ do
         if $V[peer][i] \not\in M$ then
            send $V[peer][i], C[D[peer][i]]$ to peer
         flush send buffer
   wait for all acks
   // probe other edges
   for peer ← 0 to $NP$ do
      for $i ← 0, |V[peer]|$ do
         if $V[peer][i] \not\in M$ then
            for $e ← D[peer][i]+1$ to $D[peer][i+1]$ do
               send $V[peer][i], C[e]$ to peer
         flush send buffer
```

Packed CSR Blocks

packed CSR block 0

packed CSR block NB-1

per block

old queue
new queue
visited mask

coarse-grained

old queue
new queue
visited mask

64 bits

global vertex id
rank | block | local id
Multithreading (Work in Progress)

Thread 0
- Coordinates communication and worker threads

Thread 1
- Processes received messages

Thread 2
- Processes packed CSR for one peer at a time

... (other threads)

Thread K-1
- Processes packed CSR for one peer at a time

Send buffers

Work requests

New queue

Old queue

Visited mask
Performance: Lomonosov

- 512 nodes in partition regular4, InfiniBand QDR
Performance: Lomonosov-2

- 64 nodes in R&D partition, InfiniBand FDR