Multithreaded distributed BFS with regularized communication pattern on the cluster with Angara interconnect

March 3, 2016

Artem Osipov
Alexander Daryin

www.t-platforms.com
• **Breadth-First Search (BFS)** on distributed memory systems
• Part of Graph500 benchmark

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

Local_id

0 1 2 3 ...

Rowstarts

21 7 0 13 15 42 2 27 6 27 0 0 1 0 ...

Columns

Global_id

Rank  Local_id

GraphHPC-2016
Data: CSR (rowstarts R, column C), current queue Q

function ProcessQueue(R, C, Q)
   for v in Q do
      for e in {R[v]..R[v + 1]} do
         send v, local(C[e]) to owner(C[e]))
Data: CSR (rowstarts R, column C), current queue Q

function ProcessQueue(R, C, Q)
  for v in Q do
    for p in {R[v]..R[v + 1]} do
      send v, local(C[p]) to owner(C[p]))

Reference Algorithm

www.t-platforms.com

Small message size
low bandwidth

Irregular communication pattern
connection overhead

Separate send buffer
for each peer process
large memory overhead
multiple threads - ?

Message coalescing
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)
• Partition graph data for each peer node
• Process partitions independently (and concurrently)

• **Problem:**
  • Unacceptable memory overhead when using standard CSR representation - $|\text{rowstarts}| = |V|$

• **Solution:**
  • Pack rowstarts
Packed Partitioned CSR

![Diagram of Packed Partitioned CSR](image)

- **local_id**: Indicates the local identifier for each row.
- **dst 0, dst 1, dst 2, dst 3**: Represents the destination for the rows, each with their own partitioned CSR.
- **packed rowstarts**: Represents the packed rowstarts for each destination, indicating the offset for packed rowstarts.

```
local_id | offset
---------|--------
0        | 0      
1        | 1      
2        | 2      
3        | 3      
...      | ...    
```

GraphHPC-2016
Packed CSR Algorithm

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

```plaintext
function ProcessQueue(R, C, Q)
    for \(p\) in \(\{0..NP-1\}\) do
        for \(i\) in \(\{0..|V[p]|-1\}\) do
            if \(V[p][i]\) in \(Q\) then
                for \(e\) in \(\{D[p][i]..D[p][i+1]\}\) do
                    send \(V[p][i], C[e]\) to \(p\)
```

This is a Packed CSR Algorithm designed for efficient parallel processing in High Performance Computing (HPC) applications.
• **Standard algorithm:** when sending \((u, v)\) edge for already visited \(u\) we're hoping that \(v\) is unvisited
• At some point it becomes easier to hit visited \(v\) than unvisited

• **Backward stepping algorithm:** send \((u, v)\) edge for unvisited \(u\) and hope that \(v\) is visited
• Reduce communications – use edge probing
• To increase the effectiveness of probing sort columns by degree
Data: Packed CSR (vertex indices V, row offsets D, column C), bit mask of visited vertices M, number of processes NP

function ProcessUnvisited()
// probe first edges
for p in {0..NP-1} do
  for i in {0..|V[p]|-1} do
    if V[p][i] in M then
      send V[p][i], C[D[p][i]] to peer
      flush send buffer
    wait for all acks
// probe other edges
for p in {0..NP-1} do
  for i in {0..|V[p]|-1} do
    if V[p][i] in M then
      for e in {D[p][i] + 1..D[p][i + 1]} do
        send V[peer][i], C[e] to peer
      flush send buffer
• **Basic concepts:**
  • All MPI communications go through the main thread
  • All received data is handled by dedicated thread
  • Other threads prepare data to send
  • All communications between threads go through concurrent queues (Intel Thread Building Blocks package was used)
Multithreading

Thread 0 performs MPI communication and coordinates worker threads.

Thread 1 processes received messages.

Thread 2 processes packed CSR for one dst at a time.

Thread k-1 processes packed CSR for one dst at a time.

- Send buffers
- Recv buffers
- Dst list

GraphHPC-2016
Backward stepping
Performance: “Angara K1”

GraphHPC-2016

16 Nodes

- 16x2 4 OMP
- 16x6 OMP
- 16x2 10 OMP
- 16x8
Performance: “Angara K1”

32 Nodes

- 32x1 6 OMP
- 32x4
- 32x8

GraphHPC-2016
Future optimizations

- Graph redistribution (maximal clustering per node)
- Multithreaded heavy
THANK YOU!

www.t-platforms.com