Compiler optimizations for Microservice based Cloud Applications

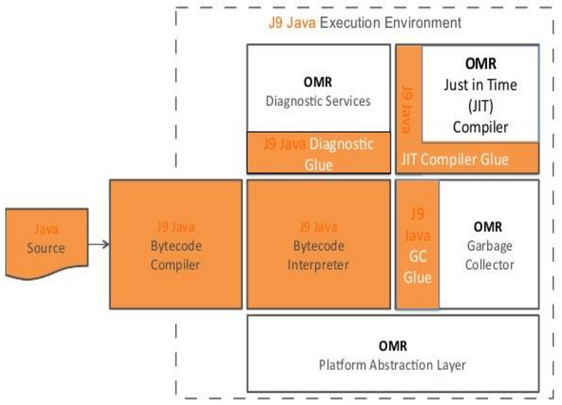
Sanil Rao, Pil Jae Jang, Pratik Fegade, Todd Mowry

INTRODUCTION

- Many applications transitioning from monolithic to cloud based
- New opportunities to optimize these micro-service applications, specifically network requests
- We performed two optimizations, shared memory and bulk loop aggregation to yield better performance

OMR

- Compiler Framework built by IBM
- Can become a part of **any** language runtime
- Exposes many features to developers like GC, threading etc.

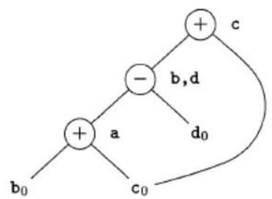


M. Stoodley, EclipseCON 2016

OMR IL

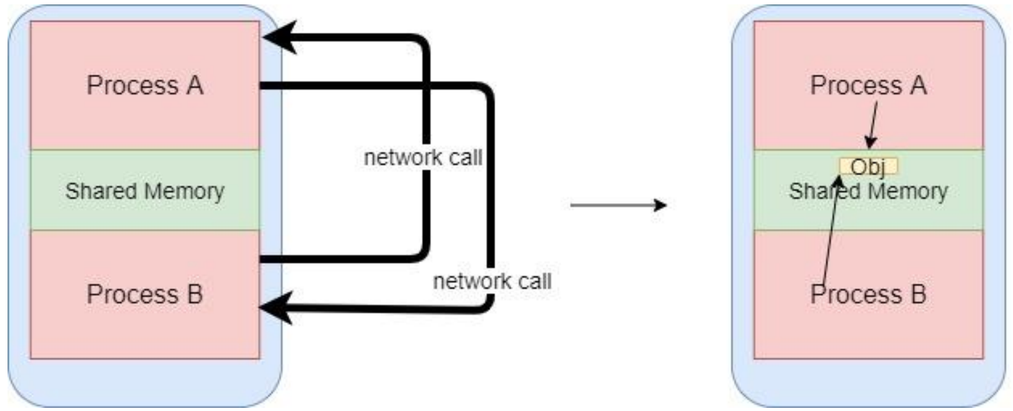
- Internal Representation of OMR using a series of DAGs
- Composed of Trees (basic blocks) and nodes (instructions)
- Program Order is determined by TreeTops a linked list of all the DAGs

```
a = b + c
b = a - d
c = b + c
d = a - d
```



Shared-Memory

- Micro services can be placed physically on the same machine within a cloud environment
- Leverage the underlying shared memory as a method of communication rather than the network



Process A

```
Int main () {
    ...
    send(socket, interesting obj, objsize);
    recv(socket, obj_data,  objsize);
    Obj  myObj = obj_data;
}
```

Process B

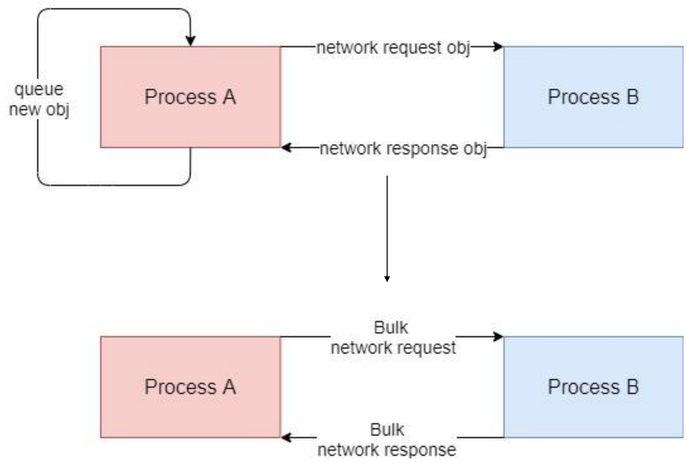
```
Int main() {
    recv(socket, interesting_obj, objsize);
    Obj o = db(interesting_obj);
    send(socket, o, osize);
}
```

Process A

```
Int main() {
    ...
    ~~send(socket, interesting obj, objsize);~~
    ~~recv(socket, obj_data,  objsize);~~
    Obj  myNewObj = db(interesting_obj);
}
```

Bulk Loop Aggregation

- One structure of requests common to microservice applications is loop based requests
- This type of request involves sending a network call for each element in a given data structure
- We instead flatten this loop of requests into a single bulk request with all the data from the data structure



Process A

```
Int main () {
    for(int i = 0; I < obj.size(); i++) {
        send(socket, obj[i], objsize);
        recv(socket, obj_data,  objsize);
        Obj  myObj = obj_data;
    }
}
```

Process A

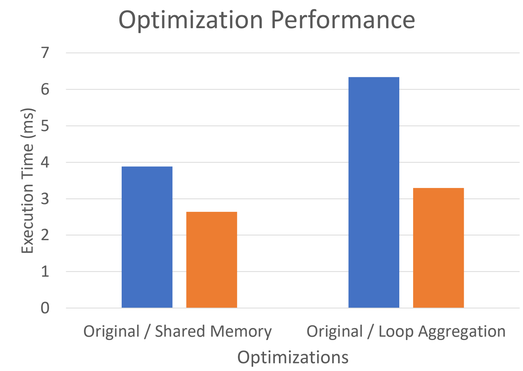
```
Int main () {
    ~~for(int i = 0; I < obj.size(); i++) {~~
        send(socket, meta_obj, objsize);
        recv(socket, meta_obj_data,  objsize);
        Obj  myObj = meta_obj_data;
    ~~}~~
}
```

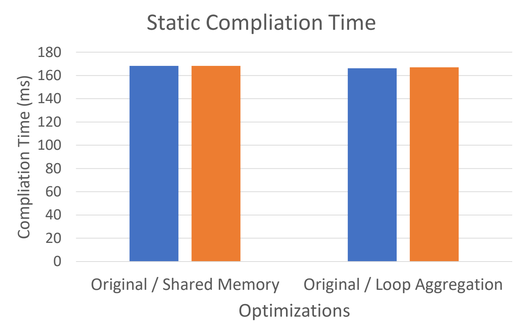
Process B

```
int main() {
    recv(socket, meta_obj_data, objsize);
    Obj d = parse_and _send (meta_obj_data);
    [MODIFIED]
}
```

Results

- We see benefits for both our optimizations
- Loop aggregation yields a higher benefit than shared memory
- Compile time stayed consistent after inclusion of our passes.
- Expect these results to be even better in real system testing rather than proof of concept.





Conclusions and Future Work

- We saw significant performance improvement with minimal effects on compilation time for our applications using these optimizations
- Shared memory had a **1.4x** improvement while loop aggregation had a **1.9x** improvement
- Future work would involve real applications that could see benefit from these types of optimizations as well as generalizing these passes for other use cases