azul

Lock Optimizations for Loops in Falcon JIT

Anna Thomas
anna@azul.com

What is Falcon?

- JIT compiler for Java based on LLVM
  - Java bytecode => assembly
  - LLVM based optimizer inside running VM
- Final tier compiler in Azul's Prime JVM

Lock Operations

- Synchronize: object is locked and then unlocked
- Expensive CPU operation
- Limits compiler optimizations

Loop Lock Coarsening

```
for (i = 1 to N) {
 synchronized(obj){
  sum += obj.x;
 }
 y++;
}
```

```
synchronized(obj){
 for (i = 1 to N) {
  sum += obj.x;
  y++;
 }
}
```

Loop Lock Coarsening

```
for (i = 1 to N) {
 synchronized(obj){
  sum += obj.x;
 }
 y++;
}
```

```
for (i = 1 to N) {
 synchronized(obj){
  sum += obj.x;
  y++;
 }
}
```

JMM: move operations (non volatile loads/stores) into critical region, not out of it

Loop Lock Coarsening

```
for (i = 1 to N) {
 synchronized(obj){
  sum += obj.x;
 }
 y++;
}
```

```
synchronized(obj){
 for (i = 1 to N) {
  sum += obj.x;
  y++;
 }
}
```

JMM: move operations (non volatile loads/stores) into critical region, not out of it

Avoid thread contention: Satisfy progress guarantees!

Loop Lock Coarsening

```
for(i=0; i<N; i++){
 synchronize(obj) {
  sum += obj.x;
 }
 y++;
}
```

Loop Lock Coarsening

```
for(i=0; i<N; i++){
 call @monitorenter(obj)
  sum += obj.x;
 call @monitorexit(obj)
 y++;
}
```

lock/unlock represented as “abstractions”: contains IR Body

Abstractions inlined at specific points in custom pipeline

Loop Lock Coarsening

Step1: Move monitorexit to latch

```
for(i=0; i<N; i++){
 call @monitorenter(obj)
  sum += obj.x;
  y++;
 call @monitorexit(obj)
}
```

Loop Lock Coarsening

Step2: Chunked original loop by ChunkSize iterations

```
for(i=0; i<N; ) {
 for(j=0; j < ChunkSize && i<N; j++,i++){
  call @monitorenter(obj)
   sum += obj.x;
   y++;
  call @monitorexit(obj)
 }
}
```

Loop Lock Coarsening

Step3: Move Monitorenter to outer loop header, monitorexit to outer loop latch.

```
for(i=0; i<N; ) {
 call @monitorenter(obj)
 for(j=0; j < ChunkSize && i<N; j++,i++){
   sum += obj.x;
   y++;
 }
 call @monitorexit(obj)
}
```

Coarsened over ChunkSize iterations. Satisfied progress guarantees

Loop Lock Coarsening

Unknown Exit condition -> MaxTripCount of ChunkSize

```
for(i=0; f(i); ) {
 call @monitorenter(obj)
 for(j=0; j < ChunkSize && f(i); j++,i++){
   sum += obj.x;
   y++;
 }
 call @monitorexit(obj)
}
```

Coarsened over ChunkSize iterations. Satisfied progress guarantees

Loop Chunking

- Move expensive operations out of chunked loop: locked fewer times
- Chunked loop can be optimized further, vectorization for example
- Other usecases when an operation executed ”every couple of iterations”
- Basis for other techniques (over chunked loop) when loop chunking is not enough

azul

Thank You!

anna@azul.com