A Pattern-based Transformation Approach to Parallelise Software Systems using a System Dependency Graph

Johanna E. Krause

27 January 2016
1. Motivation

2. Goals

3. Approach
   Mining of Candidate and Parallelisation Patterns
   Formalising Candidate Patterns
   Transformation

4. Live Demonstration

5. Evaluation

6. Conclusion and Future Work
Motivation for (Semi-)Automatic Parallelisation

- Parallel programs are mostly more performant
Motivation for (Semi-)Automatic Parallelisation

- Parallel programs are mostly more performant
- Many legacy systems would benefit from parallelisation
Motivation for (Semi-)Automatic Parallelisation

- Parallel programs are mostly more performant
- Many legacy systems would benefit from parallelisation
- Manual adjustments are time and cost consuming
Figure 1: Pattern-based detection and utilization of potential parallelism in software systems [Wulf14]
Goals

▶ Goal 1: Mining of Candidate and Parallelisation Patterns
Goals

- Goal 1: Mining of Candidate and Parallelisation Patterns
- Goal 2: Formalising Candidate Patterns
  - G2.1: Formalising as System Dependency Graph (SDG)
  - G2.2: Formalising as Cypher Match Query (CMQ)
- Goal 3: Transforming Candidate Patterns to Parallelisation Patterns
  - G3.1: Formalising as SDG
  - G3.2: Formalising as Cypher Update Query (CUQ)
- Goal 4: Evaluating the Speed-Up of the Transformed Application
Goals

- Goal 1: Mining of Candidate and Parallelisation Patterns
- Goal 2: Formalising Candidate Patterns
  - G2.1: Formalising as System Dependency Graph (SDG)
- Goal 3: Transforming Candidate Patterns to Parallelisation Patterns
  - G3.1: Formalising as SDG
  - G3.2: Formalising as Cypher Update Query (CUQ)
- Goal 4: Evaluating the Speed-Up of the Transformed Application
Goals

- **Goal 1: Mining of Candidate and Parallelisation Patterns**
- **Goal 2: Formalising Candidate Patterns**
  - G2.1: Formalising as System Dependency Graph (SDG)
  - G2.2: Formalising as Cypher Match Query (CMQ)
- **Goal 3: Transforming Candidate Patterns to Parallelisation Patterns**
- **Goal 4: Evaluating the Speed-Up of the Transformed Application**
Goals

- **Goal 1: Mining of Candidate and Parallelisation Patterns**
- **Goal 2: Formalising Candidate Patterns**
  - G2.1: Formalising as System Dependency Graph (SDG)
  - G2.2: Formalising as Cypher Match Query (CMQ)
- **Goal 3: Transforming Candidate Patterns to Parallelisation Patterns**

Johanna E. Krause
Parallelise Software Systems using an SDG
27 January 2016
Goal 1: Mining of Candidate and Parallelisation Patterns

Goal 2: Formalising Candidate Patterns
   - G2.1: Formalising as System Dependency Graph (SDG)
   - G2.2: Formalising as Cypher Match Query (CMQ)

Goal 3: Transforming Candidate Patterns to Parallelisation Patterns
   - G3.1: Formalising as SDG
Goals

- **Goal 1: Mining of Candidate and Parallelisation Patterns**
- **Goal 2: Formalising Candidate Patterns**
  - G2.1: Formalising as System Dependency Graph (SDG)
  - G2.2: Formalising as Cypher Match Query (CMQ)
- **Goal 3: Transforming Candidate Patterns to Parallelisation Patterns**
  - G3.1: Formalising as SDG
  - G3.2: Formalising as Cypher Update Query (CUQ)
Goals

- **Goal 1: Mining of Candidate and Parallelisation Patterns**
- **Goal 2: Formalising Candidate Patterns**
  - G2.1: Formalising as System Dependency Graph (SDG)
  - G2.2: Formalising as Cypher Match Query (CMQ)
- **Goal 3: Transforming Candidate Patterns to Parallelisation Patterns**
  - G3.1: Formalising as SDG
  - G3.2: Formalising as Cypher Update Query (CUQ)
- **Goal 4: Evaluating the Speed-Up of the Transformed Application**
Approach for the Thesis

Approach

Pattern Mining

1. choose prototype

Candidate Pattern

formalise pattern

Parallelisation Pattern

Parallelise

2. Cypher Match Query

MATCH (node1)
 - [:related_to] ->
 (node2)
 WHERE ...
 RETURN node1, node2

execute

3. Cypher Update Query

MATCH node...
 WHERE ...
 CREATE/DELETE ...
 RETURN node...

execute

Neo4J Database

Sequential Program

create SDG

generate

Parallelised Program

Evaluation

Resulting candidates:
1) ...
2) ...

Johanna E. Krause
Parallelise Software Systems using an SDG
27 January 2016
Independent Successive Method Calls

dataserver.connect();
eventserver.connect();
Independent Successive Method Calls

data_server.connect();
event_server.connect();

Independent For-Each Loop

for (ImportantObject o : list) {
    result = calculateSomethingForQuiteAWhile(o);
    writeResultInDatabase(result);
}
Solving Goal 1: Pattern Mining

Approach ▶ Mining of Candidate and Parallelisation Patterns

**Independent Successive Method Calls**

```java
dataserver.connect();
eventserver.connect();
```

**Independent For-Each Loop**

```java
for (ImportantObject o : list) {
    result = calculateSomethingForQuiteAWhile(o);
    writeResultInDatabase(result);
}
```

**Array Reduction**

```java
int sum = 0;
for (int i = 0; i < array.length; i++) {
    sum = sum + array[i];
}
```

[Molitorisz12, Mattson04]
Solving Goal 1: Pattern Mining

Approach ▶ Mining of Candidate and Parallelisation Patterns

Independent Successive Method Calls

dataserver.connect();
eventserver.connect();

Independent For-Each Loop

for (ImportantObject o : list) {
    result = calculateSomethingForQuiteAWhile(o);
    writeResultInDatabase(result);
}

Array Reduction

int sum = 0;
for (int i = 0; i < array.length; i++) {
    sum = sum + array[i];
}

[Molitorisz12, Mattson04]
Solving Goal 2: Pattern Matching

SDG of Candidate Pattern

Approach ▶ Formalising Candidate Patterns

Diagram:

- Method
- MethodCall
- MethodCall/Assignment
- avDur > 200 ms
- calls
- aggregated_field_reads
- aggregated_field_writes
- aggregated_calls
- ...
MATCH (m1:MethodCall)
  −[:CONTROL_FLOW*1..5] − >
  (m2:MethodCall)
RETURN collect(DISTINCT id(m1))
MATCH (m1:MethodCall)
  −[:CONTROL_FLOW*1..5] − >
  (m2:MethodCall)
WHERE m1.avgDurInMs > 200 AND m2.avgDurInMs > 200
RETURN collect(DISTINCT id(m1))
Solving Goal 2: Pattern Matching
Restriction: Minimum Average Duration

Approach ▶ Formalising Candidate Patterns

MATCH (m1:MethodCall)
    −[:CONTROL_FLOW*1..5] −>
    (m2:MethodCall)
WHERE m1.avgDurInMs > 200 AND m2.avgDurInMs > 200
RETURN collect(DISTINCT id(m1))

runtime information configurable
MATCH (m1:MethodCall)
    −[cfs:CONTROL_FLOW*1..5] −> (m2:MethodCall)
WHERE m1.avgDurInMs > 200 AND m2.avgDurInMs > 200
AND none(cf IN cfs WHERE has(cf.case))
RETURN collect(DISTINCT id(m1))
MATCH (m1:MethodCall)
    −[cfs:CONTROL_FLOW*1..5] −>
    (m2:MethodCall)
WHERE m1.avgDurInMs > 200 AND m2.avgDurInMs > 200
AND none(cf IN cfs WHERE has(cf.case))
RETURN collect(DISTINCT id(m1))

boolean isAvailable = isProductAvailable();
if(isAvailable){
    processOrder();
}
MATCH (m1: MethodCall)
    −[cfs :CONTROL_FLOW*1..5] − >
    (m2: MethodCall)
WHERE m1.avgDurInMs > 200 AND m2.avgDurInMs > 200
AND none (cf IN cfs WHERE has (cf.case))
AND NOT (m1) −[:DATA_FLOW*1..5] − > (m2)
RETURN collect (DISTINCT id (m1))
MATCH (m1:MethodCall)
  −[cfs:CONTROL_FLOW*1..5] -> (m2:MethodCall)
WHERE m1.avgDurInMs > 200 AND m2.avgDurInMs > 200
AND none(cf IN cfs WHERE has(cf.case))
AND NOT (m1) −[:DATA_FLOW*1..5] -> (m2)
RETURN collect(DISTINCT id(m1))

```java
int stock = materialInStock();
boolean enough = isEnoughInStock(stock);
makeOrders(enough);
```
MATCH (d1:Method) ←[:CALLS]→ (m1:MethodCall) ←[cfs:CONTROL_FLOW\texttt{\*}1..5]→ (m2:MethodCall) ←[:CALLS]→ (d2:Method)
WHERE m1.avgDurInMs > 200 AND m2.avgDurInMs > 200
AND none (cf IN cfs WHERE exists (cf.case))
AND NOT (m1) ←[:DATA\_FLOW\texttt{\*}1..5]→ (m2)
AND NOT (m1) ←[:DATA\_FLOW]→ (:Field) ←[:DATA\_FLOW]→ (m2)
AND d1.isParallelisable=true
AND d2.isParallelisable=true
RETURN collect(DISTINCT id(m1))
Solving Goal 2: Pattern Matching
Restriction: No Modification of Concurrently Accessed Fields
Approach ▶ Formalising Candidate Patterns

MATCH (d1:Method) ←[:CALLS]→ (m1:MethodCall) ←[cfs :CONTROL_FLOW*1..5]→ (m2:MethodCall) ←[:CALLS]→ (d2:Method)
WHERE m1.avgDurInMs > 200 AND m2.avgDurInMs > 200
AND none (cf IN cfs WHERE exists (cf.case))
AND NOT (m1) ←[:DATA_FLOW*1..5]→ (m2)
AND NOT (m1) ←[:DATA_FLOW]→ (:Field) ←[:DATA_FLOW]→ (m2)
AND d1.isParallelisable=true
AND d2.isParallelisable=true
RETURN collect (DISTINCT id (m1))
Solving Goal 2: Pattern Matching

Restriction: No Dependency From 1. Statement to Intermediate Ones

Approach ▶ Formalising Candidate Patterns

MATCH (d1: Method) ←[:CALLS]→
     (m1: MethodCall) ←[cfs: CONTROL_FLOW*1..5]→ (m2: MethodCall)
     ←[:CALLS]→ (d2: Method)
WHERE m1.avgDurationInMs > 200 AND m2.avgDurationInMs > 200
WITH m1, m2, d1, d2, cfs
MATCH path = (m1) ←[:CONTROL_FLOW*1..5]→ (m2)
WITH m1, m2, d1, d2, cfs, filter (intermediateNode IN nodes(path)
     WHERE intermediateNode <> m1
     AND intermediateNode <> m2)
     AS intermediateNodes
WHERE
    NOT (m1) ←[:DATA_FLOW*1..5]→ (m2)
    AND none(cf IN cfs WHERE exists(cf.case))
    AND NOT (m1) ←[:DATA_FLOW]→ (:Field) ←[:DATA_FLOW]→ (m2)
    AND d1.isParallelisable=true
    AND d2.isParallelisable=true

AND all(node IN intermediateNodes
    WHERE
        NOT (m1) ←[:DATA_FLOW]→ (node)
        AND (NOT node:MethodCall
            OR all(pathcall IN ((node) ←[:CALLS]→ ()
                WHERE all(call IN rels(pathcall)
                    WHERE endNode(call).isParallelisable=true
                    OR endNode(call):Constructor))))
RETURN collect(DISTINCT id(m1))
Solving Goal 2: Pattern Matching
Extension: No Modification of Concurrently Accessed Fields
Approach ▶ Formalising Candidate Patterns

... AND NOT d1 overridden=true AND NOT d2 overridden=true
AND (d1 isParallelisable=true
    OR NOT (d1) − [AGGREGATED_FIELD_WRITE
                     |AGGREGATED_CALLS* ]→ (:Field) ― (d2))
AND (d2 isParallelisable=true
    OR NOT (d2) − [AGGREGATED_FIELD_WRITE
                     |AGGREGATED_CALLS* ]→ (:Field) ― (d1)) ...

▶ less restrictive: allow modification of fields except concurrently accessed ones
▶ Attention: handle overridden methods separately!
⇒ see details in thesis
Master Worker Pattern:

- Usage of nested Callables and Futures
  - Enables return value
  - Enables exception handling
- Organisation with Java’s ExecutorsService (thread pool)

no Java 8 support (because of Soot)

Source: https://github.com/Sable/soot/issues/394
Solving Goal 3: Transformation
Example of SDGs – Before and After

Approach ▶ Transformation

Candidate Pattern

Parallelisation Pattern
Solving Goal 3: Transformation

Mission

Approach ▶ Transformation

- Add new nodes according to Soot representation
- Add new control flows and hierarchy
- Remove unused control flows
- New variable/class names (variable scope)
- Exception handling
Solving Goal 3: Transformation

Mission

Approach ▶ Transformation

- Add new nodes according to Soot representation
- Add new control flows and hierarchy
- Remove unused control flows
- New variable/class names (variable scope)
- Exception handling

Simplification/Optimisation:
No attention to data flows, instead new Soot run
Solving Goal 3: Transformation

Transformation Implementation

Approach

- Implementation in Java
- Cypher queries (from String)
- Neo4J Java API

Advantages:
- Reusability of the queries
- Dynamic build of queries
- Temporary storing of nodes
Solving Goal 3: Transformation

Transformation Implementation

Approach ▶ Transformation

- Implementation in Java
- Cypher queries (from String)
- Neo4J Java API

Advantages:
- Reusability of the queries
- Dynamic build of queries
- Temporary storing of nodes
  ⇒ Comfortable handling of relationships
Live Demonstration
Evaluation of the speed up not yet possible:
Evaluation of the speed up not yet possible:

- Generation of Java source code from the Neo4J SDG is very complex
Evaluation of the speed up not yet possible:

- Generation of Java source code from the Neo4J SDG is very complex
  - Try-catch blocks
  - Throw exceptions
  - Loop
Evaluation of the speed up not yet possible:

- Generation of Java source code from the Neo4J SDG is very complex
  - Try-catch blocks
  - Throw exceptions
  - Loop

⇒ Approach differently evaluated
Evaluation

- Feasibility of the approach
- Quantitative occurrence of the candidate patterns
- Extendibility of the approach
Yes, we can!
We successfully transformed three prototypes :)
Checkstyle and Findbugs

<table>
<thead>
<tr>
<th>#</th>
<th>Checkstyle</th>
<th>Findbugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall nodes in SDG</td>
<td>83619</td>
<td>140875</td>
</tr>
<tr>
<td>All classes in SDG</td>
<td>942</td>
<td>1357</td>
</tr>
<tr>
<td>All Methods in SDG (without constructors)</td>
<td>8759</td>
<td>11983</td>
</tr>
<tr>
<td>Classes in app</td>
<td>762</td>
<td>1160</td>
</tr>
<tr>
<td>Methods in app (without constructors)</td>
<td>6772</td>
<td>9392</td>
</tr>
<tr>
<td>Overridden methods in app</td>
<td>754</td>
<td>505</td>
</tr>
</tbody>
</table>
### Independent Successive Method Calls Pattern

<table>
<thead>
<tr>
<th>#</th>
<th>Checkstyle</th>
<th>Findbugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes in app</td>
<td>762</td>
<td>1160</td>
</tr>
<tr>
<td>Methods in app (without constructors)</td>
<td>6772</td>
<td>9392</td>
</tr>
<tr>
<td>Read-only methods</td>
<td>963</td>
<td>1953</td>
</tr>
<tr>
<td>Method calls</td>
<td>20664</td>
<td>39699</td>
</tr>
<tr>
<td>Read-only-method calls</td>
<td>1208</td>
<td>4078</td>
</tr>
<tr>
<td>Successive method calls</td>
<td>12870</td>
<td>29595</td>
</tr>
<tr>
<td>Successive read-only method calls</td>
<td>26</td>
<td>352</td>
</tr>
<tr>
<td>Independent successive method calls</td>
<td>551</td>
<td>–</td>
</tr>
</tbody>
</table>
## Independent Successive Method Calls Pattern

<table>
<thead>
<tr>
<th>#</th>
<th>Checkstyle</th>
<th>Findbugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes in app</td>
<td>762</td>
<td>1160</td>
</tr>
<tr>
<td>Methods in app (without constructors)</td>
<td>6772</td>
<td>9392</td>
</tr>
<tr>
<td>Read-only methods</td>
<td>963</td>
<td>1953</td>
</tr>
<tr>
<td>Method calls</td>
<td>20664</td>
<td>39699</td>
</tr>
<tr>
<td>Read-only-method calls</td>
<td>1208</td>
<td>4078</td>
</tr>
<tr>
<td>Successive method calls</td>
<td>12870</td>
<td>29595</td>
</tr>
<tr>
<td>Successive read-only method calls</td>
<td>26</td>
<td>352</td>
</tr>
<tr>
<td>Independent successive method calls</td>
<td>551</td>
<td>–</td>
</tr>
</tbody>
</table>

- without runtime information constraints
- branches excluded
- no allowance of ‘isParallelisable=true’ for more flexibility
Independent For-Each Loop

<table>
<thead>
<tr>
<th></th>
<th>Checkstyle</th>
<th>Findbugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes in app</td>
<td>762</td>
<td>1160</td>
</tr>
<tr>
<td>Methods in app (without constructors)</td>
<td>6772</td>
<td>9392</td>
</tr>
<tr>
<td>For-Each Loops</td>
<td>97</td>
<td>234*</td>
</tr>
<tr>
<td>Read-only For-Each Loops</td>
<td>16</td>
<td>75*</td>
</tr>
<tr>
<td>Independent For-Each Loops</td>
<td>39</td>
<td>103*</td>
</tr>
</tbody>
</table>

* loop body size restricted to a maximum of 30 statements
### Array Reduction Pattern

<table>
<thead>
<tr>
<th></th>
<th>Checkstyle</th>
<th>Findbugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes in app</td>
<td>762</td>
<td>1160</td>
</tr>
<tr>
<td>Methods in app (without constructors)</td>
<td>6772</td>
<td>9392</td>
</tr>
<tr>
<td>Array length operation</td>
<td>717</td>
<td>636</td>
</tr>
<tr>
<td>Array access operation</td>
<td>614</td>
<td>982</td>
</tr>
<tr>
<td>Array assignments</td>
<td>2424</td>
<td>536</td>
</tr>
<tr>
<td>Array reduction</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Pros

- Modular because of Java
- Reusability of queries
- Semi-automatism easily extendible
- Configurable, e.g. runtime constraints
- New candidate and parallelisation patterns can be designed with the help of existing utility classes

Cons

- When SDG changes, CMQs and transformation have to be adjusted
  - Maintenance effort, e.g. for Java 8 support
Conclusion:

- Successful implementation of 3 patterns
- Powerful, but complex approach

Future Work:

- Generate Java code from SDG for performance evaluation
- Add runtime information
- Implement additional candidate and parallelisation patterns
Conclusion and Future Work

Conclusion:
- Successful implementation of 3 patterns
- Powerful, but complex approach

Future Work:
- Generate Java code from SDG for performance evaluation
- Add runtime information
- Implement additional candidate and parallelisation patterns
References

Conclusion and Future Work


Approach for the Thesis

Conclusion and Future Work

**Pattern Mining**

1. choose prototype

   **Candidate Pattern**

   2. formalise pattern

   **Parallelisation Pattern**

2. Cypher Match Query

   MATCH (node1)
   -[:related_to]->
   (node2)
   WHERE ...
   RETURN node1, node2

3. Cypher Update Query

   MATCH node...
   WHERE ...
   CREATE/DELETE ...
   RETURN node...

4. Evaluation

   - Parallelised Program
   - Evaluation
   - Resulting candidates: 1) ...
   - Parallelise
   - execute

Sequential Program

create SDG

Neo4J Database

JAVA

JAVA

0.

generate

Parallelised Program

Johanna E. Krause

Parallelise Software Systems using an SDG

27 January 2016 31 / 37
MATCH (m: Method) ←[:CONTAINS_METHOD]− (classOrInterface) ←[:EXTENDS|IMPLEMENTS*1..]− (subclass) ←[:CONTAINS_METHOD] (method: Method)
WHERE m.displayname = method.displayname
SET m.overridden = true
**In theory:**

MATCH (m:Method)
WHERE
    NOT (exists(m.overridden) OR m.overridden <> true)
    AND NOT (m) -[:AGGREGATED_FIELD_WRITE|AGGREGATED_CALLS*] -
WITH m
SET m.isReadOnly=true
In our ‘capped’ SDG:

1. Cypher-Query:

MATCH (m:Method)
WHERE
  m.origin = 'APP'
  AND (NOT exists(m.overridden) OR m.overridden <> true)
  AND NOT (m) -> [:AGGREGATED_FIELD_WRITE] -> (:Field)
  AND (NOT (m) -> [:AGGREGATED_CALLS] -> (:Method))
WITH m
SET m.isReadOnly=true
Solving Goal 2: Pattern Matching
Introducing the attributes isReadOnly

Conclusion and Future Work

2.-x. Cypher query

MATCH (mRO:Method) ←[:AGGREGATED_CALLS]→ (m:Method)
WHERE
  mRO.isReadOnly=true
AND NOT EXISTS(m.isReadOnly)
AND (NOT EXISTS(m.overridden) OR m.overridden <> true)
AND NOT (m) ←[:AGGREGATED_FIELD_WRITE]→ (:Field)
AND (all(path IN ((m) ←[:AGGREGATED_CALLS]→ (:Method)))
  WHERE all(method IN nodes(path)
    WHERE m = method
    OR method.isReadOnly=true)))

WITH m
SET m.isReadOnly=true
int nProcessors = ParallelisationUtil.NUMBER_OF_PROCESSORS;
java.util.concurrent.ExecutorService pool = java.util.concurrent.Executors
    .newFixedThreadPool(nProcessors);

DataStreamCallable callable1 = new ConnectionSetup.DataStreamCallable(dataSC);
java.util.concurrent.Future<Object> future1 = pool.submit(callable1);

DataStreamCallable callable2 = new ConnectionSetup.EventDataStreamCallable(eventSC);
java.util.concurrent.Future<Object> future2 = pool.submit(callable2);

try {
    future1.get();
    future2.get();
} catch (InterruptedException e) {
}
} catch (java.util.concurrent.ExecutionException e) {
    Throwable cause = e.getCause();
    if (cause instanceof Error) {
        throw (Error) cause;
    }
    if (cause instanceof RuntimeException) {
        throw (RuntimeException) cause;
    }
}
pool.shutdown();

...
private static class DataSCConnectCallable implements Callable<Void> {
    private IDataServerConnection dataSC;

    public DataSCConnectCallable(IDataServerConnection dataSC) {
        super();
        this.dataSC = dataSC;
    }

    @Override
    public Void call() throws Exception {
        dataSC.connect();
        return null;
    }
}

private static class EventSCConnectCallable implements Callable<Void> {
    private IEventServerConnection eventSC;

    public EventSCConnectCallable(IEventServerConnection eventSC) {
        super();
        this.eventSC = eventSC;
    }

    @Override
    public Void call() {
        eventSC.connect();
        return null;
    }
}