Translating imperative code to MapReduce

Cosmin Rădoi [1] Stephen J. Fink [2]
Rodric Rabbah [2] Manu Sridharan [3]

[1] University of Illinois

[2] IBM T.J. Watson Research Center

[3] Samsung Research America

background: What is MapReduce?

Jeffrey Dean and Sanjay Ghemawat.

MapReduce: simplified data processing on large clusters. OSDI'04

background: What is MapReduce?

simple programing model for processing big data on a cluster

Jeffrey Dean and Sanjay Ghemawat.

MapReduce: simplified data processing on large clusters. OSDI'04

background: What is MapReduce?

simple programing model for processing big data on a cluster

advantages:

- fault-tolerance
- elastic scaling
- integration with distributed file systems

Jeffrey Dean and Sanjay Ghemawat.

MapReduce: simplified data processing on large clusters. OSDI'04

background: What is MapReduce?

simple programing model for processing big data on a cluster

advantages:

- fault-tolerance
- elastic scaling
- integration with distributed file systems
- popular ecosystem — many good tools and services

Jeffrey Dean and Sanjay Ghemawat.

MapReduce: simplified data processing on large clusters. OSDI'04

Why translate automatically to MapReduce?

- although simple, MapReduce is not easy

Why translate automatically to MapReduce?

- although simple, MapReduce is not easy
- reduce cost of retargeting legacy imperative code

Why translate automatically to MapReduce?

- although simple, MapReduce is not easy
- reduce cost of retargeting legacy imperative code
- allow developers to concentrate on familiar imperative sequential code

WordCount

example: MapReduce WordCount

doc1

cat dog dog

doc2

cat cat

example: MapReduce WordCount

doc1

cat dog dog

doc2

cat cat

MAP ↓ ↓

⟨cat, 1⟩
⟨dog, 1⟩
⟨dog, 1⟩

⟨cat, 1⟩
⟨cat, 1⟩

example: MapReduce WordCount

	doc1	doc2
	cat dog dog	cat cat
MAP	↓	↓
	⟨cat, 1⟩ ⟨dog, 1⟩ ⟨dog, 1⟩	⟨cat, 1⟩ ⟨cat, 1⟩

example: MapReduce WordCount

doc1	doc2
cat dog dog	cat cat

MAP ↓ ↓

$\langle$cat, 1$\rangle$
$\langle$dog, 1$\rangle$
$\langle$dog, 1$\rangle$

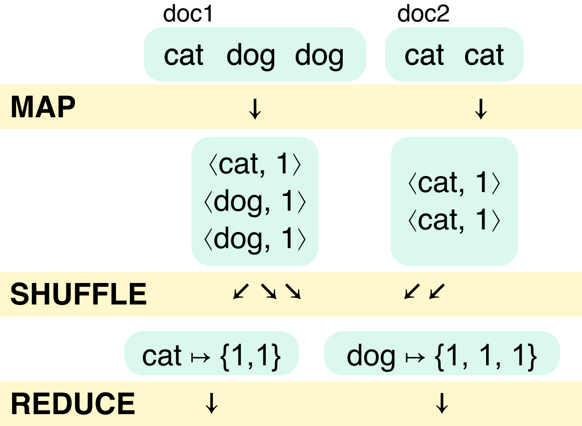
$\langle$cat, 1$\rangle$
$\langle$cat, 1$\rangle$

SHUFFLE ↙ ↘ ↘ ↙ ↙

cat $\mapsto$ {1,1}

dog $\mapsto$ {1, 1, 1}

example: MapReduce WordCount



example: MapReduce WordCount

doc1: cat dog dog

doc2: cat cat

MAP $\downarrow$ $\downarrow$

doc1: $\langle$cat, 1$\rangle$ $\langle$dog, 1$\rangle$ $\langle$dog, 1$\rangle$

doc2: $\langle$cat, 1$\rangle$ $\langle$cat, 1$\rangle$

SHUFFLE $\swarrow$ $\searrow$ $\searrow$ $\swarrow$ $\swarrow$

cat $\mapsto$ {1,1}

dog $\mapsto$ {1, 1, 1}

REDUCE $\downarrow$ $\downarrow$

cat $\mapsto$ 2

dog $\mapsto$ 3

example

```java
Map<String,Integer> count = new HashMap<>();

for (int i = 0; i < docs.size(); i++) {
  String[] words = tokenize(docs.get(i));
  for (int j = 0; j < words.length; j++) {
    String word = words[j];
    Integer prev = count.get(word);
    if (prev == null) prev = 0;
    count.put(word, prev + 1);
  }
}
```

example: imperative WordCount

```
Map<String,Integer> count = new HashMap<>();

for (int i = 0; i < docs.size(); i++) {
  String[] words = tokenize(docs.get(i));
  for (int j = 0; j < words.length; j++) {
    String word = words[j];
    Integer prev = count.get(word);
    if (prev == null) prev = 0;
    count.put(word, prev + 1);
  }
}
```

example: imperative WordCount

```
Map<String,Integer> count = new HashMap<>();

for (int i = 0; i < docs.size(); i++) {
  String[] words = tokenize(docs.get(i));
  for (int j = 0; j < words.length; j++) {
    String word = words[j];
    Integer prev = count.get(word);
    if (prev == null) prev = 0;
    count.put(word, prev + 1);
  }
}
```

example: imperative WordCount

```
Map<String,Integer> count = new HashMap<>();

for (int i = 0; i < docs.size(); i++) {
  String[] words = tokenize(docs.get(i));
  for (int j = 0; j < words.length; j++) {
    String word = words[j];
    Integer prev = count.get(word);
    if (prev == null) prev = 0;
    count.put(word, prev + 1);
  }
}
```

Example - imperative WordCount

```java
Map<String,Integer> count = new HashMap<>();

for (int i = 0; i < docs.size(); i++) {
  String[] words = tokenize(docs.get(i));
  for (int j = 0; j < words.length; j++) {
    String word = words[j];
    Integer prev = count.get(word);
    if (prev == null) prev = 0;
    count.put(word, prev + 1);
  }
}
```

example: imperative WordCount

```java
Map<String,Integer> count = new HashMap<>();

for (int i = 0; i < docs.size(); i++) {
  String[] words = tokenize(docs.get(i));
  for (int j = 0; j < words.length; j++) {
    String word = words[j];
    Integer prev = count.get(word);
    if (prev == null) prev = 0;
    count.put(word, prev + 1);
  }
}
```

example: imperative WordCount ⇒ MapReduce

```java
Map<String,Integer> count = new HashMap<>();

for (int i = 0; i < docs.size(); i++) {
  String[] words = tokenize(docs.get(i));
  for (int j = 0; j < words.length; j++) {
    String word = words[j];
    Integer prev = count.get(word);
    if (prev == null) prev = 0;
    count.put(word, prev + 1);
  }
}
```

$\Downarrow$

```scala
docs
  .flatMap({ case (i, doc) => tokenize(doc) })
  .map({ case (j, word) => (word, 1) })
  .reduceByKey({ case (c1, c2) => c1 + c2 })
```

the MAP

```
Map<String,Integer> count = new HashMap<>();

for (int i = 0; i < docs.size(); i++) {
  String[] words = tokenize(docs.get(i));
  for (int j = 0; j < words.length; j++) {
    String word = words[j];
    Integer prev = count.get(word);
    if (prev == null) prev = 0;
    count.put(word, prev + 1);
  }
}
```

$\Downarrow$

```
docs
  .flatMap({ case (i, doc) => tokenize(doc) })
  .map({ case (j, word) => (word, 1) })
  .reduceByKey({ case (c1, c2) => c1 + c2 })
```

```
Map<String,Integer> count = new HashMap<>();

for (int i = 0; i < docs.size(); i++) {
  String[] words = tokenize(docs.get(i));
  for (int j = 0; j < words.length; j++) {
    String word = words[j];
    Integer prev = count.get(word);
    if (prev == null) prev = 0;
    count.put(word, prev + 1);
  }
}
```

$\Downarrow$

```
docs
  .flatMap({ case (i, doc) => tokenize(doc) })
  .map({ case (j, word) => (word, 1) })
  .reduceByKey({ case (c1, c2) => c1 + c2 })
```

the REDUCE

```
Map<String,Integer> count = new HashMap<>();

for (int i = 0; i < docs.size(); i++) {
  String[] words = tokenize(docs.get(i));
  for (int j = 0; j < words.length; j++) {
    String word = words[j];
    Integer prev = count.get(word);
    if (prev == null) prev = 0;
    count.put(word, prev + 1);
  }
}
```

$\Downarrow$

```
docs
  .flatMap({ case (i, doc) => tokenize(doc) })
  .map({ case (j, word) => (word, 1) })
  .reduceByKey({ case (c1, c2) => c1 + c2 })
```

MapReduce generated by MOLD

```
Map<String,Integer> count = new HashMap<>();

for (int i = 0; i < docs.size(); i++) {
  String[] words = tokenize(docs.get(i));
  for (int j = 0; j < words.length; j++) {
    String word = words[j];
    Integer prev = count.get(word);
    if (prev == null) prev = 0;
    count.put(word, prev + 1);
  }
}
```

$\Downarrow$

```
docs
  .flatMap({ case (i, doc) => tokenize(doc) })
  .map({ case (j, word) => (word, 1) })
  .reduceByKey({ case (c1, c2) => c1 + c2 })
```

MapReduce generated by MOLD

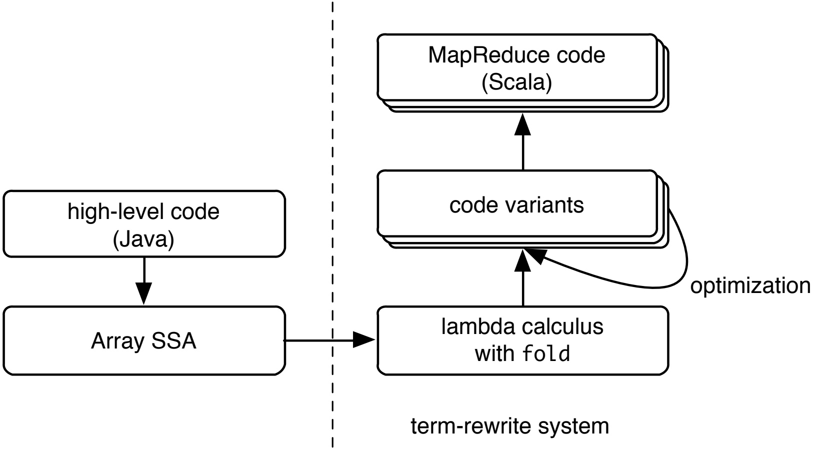
```
Map<String,Integer> count = new HashMap<>();

for (int i = 0; i < docs.size(); i++) {
  String[] words = tokenize(docs.get(i));
  for (int j = 0; j < words.length; j++) {
    String word = words[j];
    Integer prev = count.get(word);
    if (prev == null) prev = 0;
    count.put(word, prev + 1);
  }
}
```

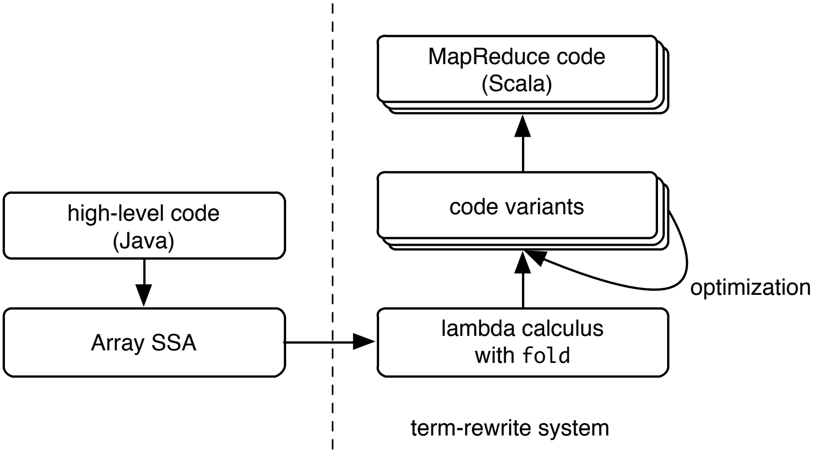
$\Downarrow$ **MOLD**

```
docs
  .flatMap({ case (i, doc) => tokenize(doc) })
  .map({ case (j, word) => (word, 1) })
  .reduceByKey({ case (c1, c2) => c1 + c2 })
```

high-level "**How?**"

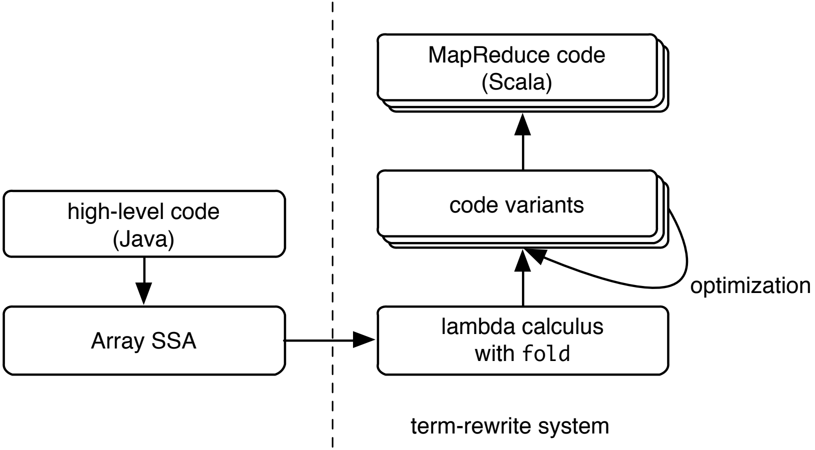


high-level “**How?**”



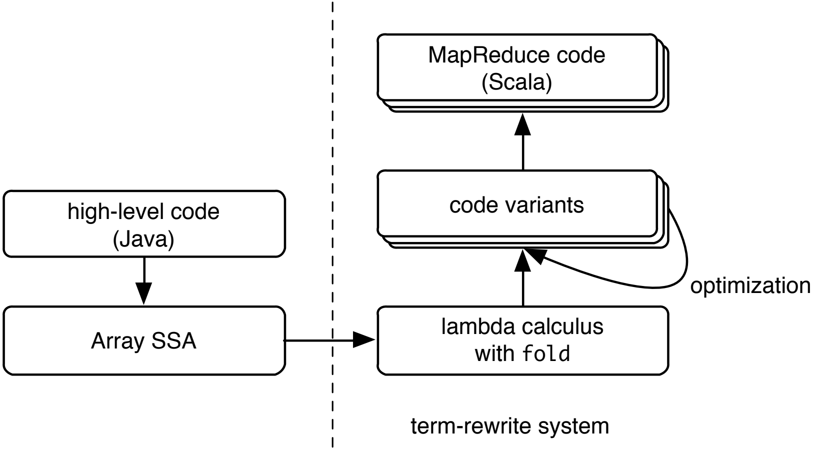
Kathleen Knobe and Vivek Sarkar. Array SSA form and its use in parallelization. POPL’98

high-level "**How?**"

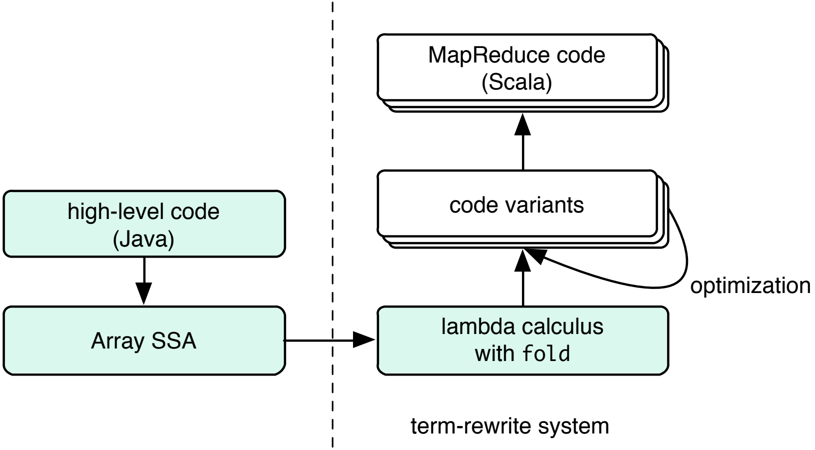


Andrew W. Appel. SSA is Functional Programming. '98
Richard Kelsey. A Correspondence between Continuation Passing Style and Static Single Assignment Form. '95

high-level “**How?**”



high-level "**How?**"



imperative ⇒ unoptimized functional

```
for (int i = 0; i < docs.size(); i++) {
  String[] words = tokenize(docs.get(i));
  for (int j = 0; j < words.length; j++) {
    String word = words[j];
    Integer prev = count.get(word);
    if (prev == null) prev = 0;
    count.put(word, prev + 1);
  }
}
```

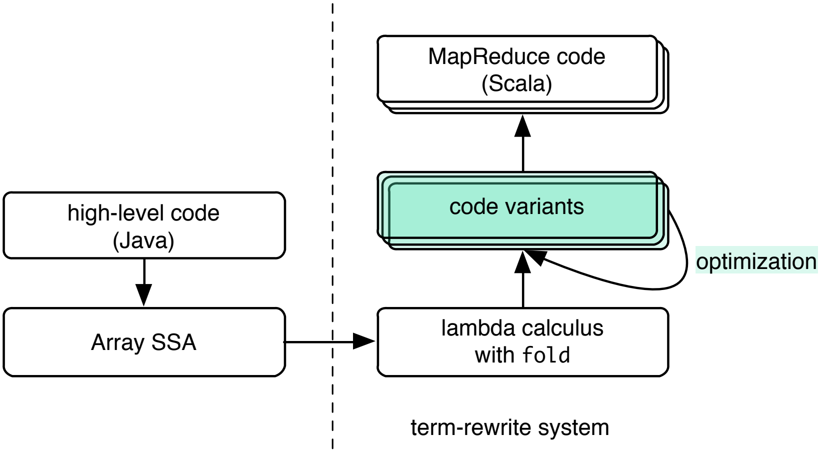
$\Downarrow$

```
words.fold(count){ (count, word) =>
    count.update(word, count(word) + 1)
}
```

how to parallelize this?

```
words.fold(count){ (count, word) =>
    count.update(word, count(word) + 1)
}
```

high-level “**How?**”



trying a `fold` ⇒ `map` transformation ...

```
words.fold(count){ (count, word) =>
    count.update(word, count(word) + 1)
}
```

trying a `fold` ⇒ `map` transformation ...

```
words.fold(count){ (count, word) =>
    count.update(word, count(word) + 1)
}
```

...does not work:
distinct `fold` iterations can write to the same key in `count`

trying a `fold` ⇒ `groupBy` transformation ...

```
words.fold(count){ (count, word) =>
    count.update(word, count(word) + 1)
}
```

fold ⇒ groupBy

```
words.fold(count){ (count, word) =>
    count.update(word, count(word) + 1)
}
```

$\Downarrow$

```
words.groupBy(word => word).map { (word, list) =>
    list.fold(count(word)) { (sum, elem) => sum + 1  }
}
```

```
words.fold(count){ (count, word) =>
    count.update(word, count(word) + 1)
}
```

$$\Downarrow$$

```
words.groupBy(word => word).map { (word, list) =>
    list.fold(count(word)) { (sum, elem) => sum + 1  }
}
```

groupBy ≡ SHUFFLE

generic `fold` ⇒ `groupBy`

```
words.fold(count){ (count, word) =>
    count.update(word, count(word) + 1)
}
```

$$\Downarrow$$

```
words.groupBy(word => word).map { (word, list) =>
    list.fold(count(word)) { (sum, elem) => sum + 1  }
}
```

`fold` ⇒ `groupBy`

```
D.fold(A) { (a, k) => a.update(I, E) }
```

$$\Downarrow$$

```
D.groupBy( k => I ).map { (i, l) =>
   l.fold(A(i)) { (r, k) => E [ r / a(I) ] }}
```

$I \not\ni a$

$E \not\ni A(\neq I)$

tranformation rules

fold ⇒ groupBy

$$\frac{\texttt{D.fold(A) \{ (a, k) => a.update(I, E) \}}}{\begin{array}{l}\texttt{D.groupBy(k => I).map \{ (i, l) =>} \\ \quad \texttt{l.fold(A(i)) \{ (r, k) => E [r / a(I)] \}\}}\end{array}} \qquad \begin{array}{l}\texttt{I} \not\ni \texttt{a} \\ \texttt{E} \not\ni \texttt{A}(\neq\texttt{I})\end{array}$$

tranformation rules

fold ⇒ groupBy

```
D.fold(A) { (a, k) => a.update(I, E) }
```

$$\Downarrow$$

```
D.groupBy( k => I ).map { (i, l) =>
   l.fold(A(i)) { (r, k) => E [ r / a(I) ] }}
```

$\mathtt{I} \not\ni \mathtt{a}$

$\mathtt{E} \not\ni \mathtt{A}(\neq \mathtt{I})$

fold ⇒ map

```
D.fold(A) { (a, k) => a.update(I, E) }
```

$$\Downarrow$$

```
A.map { (k, v) => E [ v / a(k) ] }
```

$\mathtt{D} = \mathtt{A.keys}$

$\mathtt{I} = \mathtt{k}$

$\mathtt{E} \not\ni \mathtt{A}(\neq \mathtt{I})$

tranformation rules

fold ⇒ groupBy

```
D.fold(A) { (a, k) => a.update(I, E) }
```

$$\Downarrow$$

```
D.groupBy( k => I ).map { (i, l) =>
   l.fold(A(i)) { (r, k) => E [ r / a(I) ] }}
```

$I \not\ni a$

$E \not\ni A(\neq I)$

fold ⇒ map

```
D.fold(A) { (a, k) => a.update(I, E) }
```

$$\Downarrow$$

```
A.map { (k, v) => E [ v / a(k) ] }
```

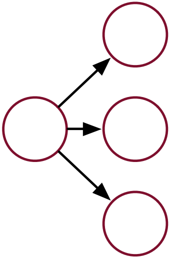
$D = A.keys$

$I = k$

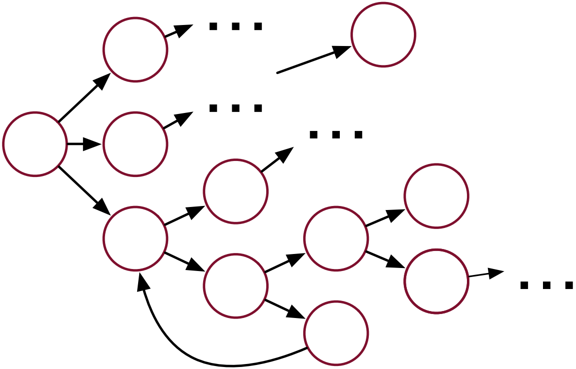
$E \not\ni A(\neq I)$

... 16 more ...

at each step, MOLD can apply any of several transfomation rules

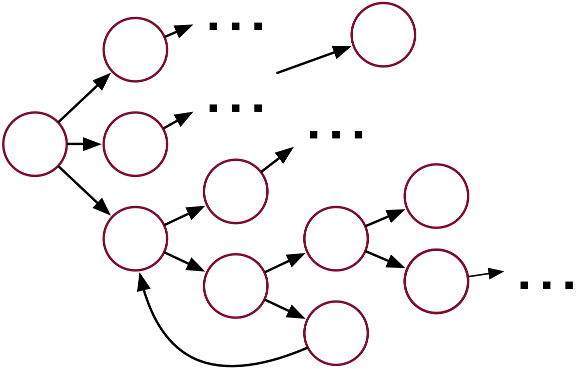


the system is not confluent, nor terminating



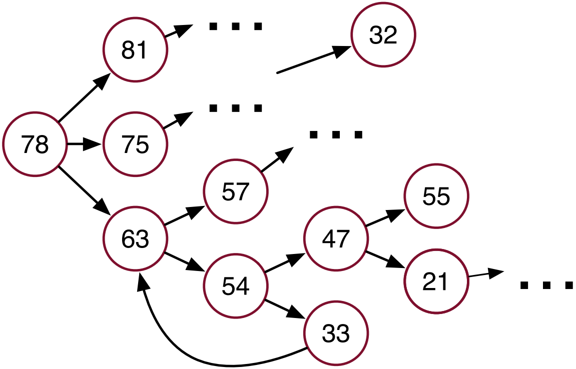
Program variant exploration

⇒ exploration/search



Program variant exploration

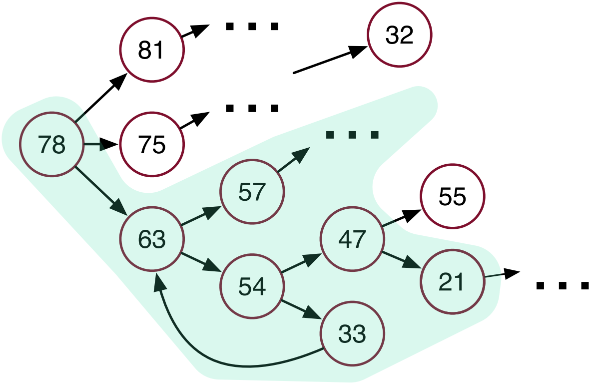
MOLD attaches a cost to each program variant



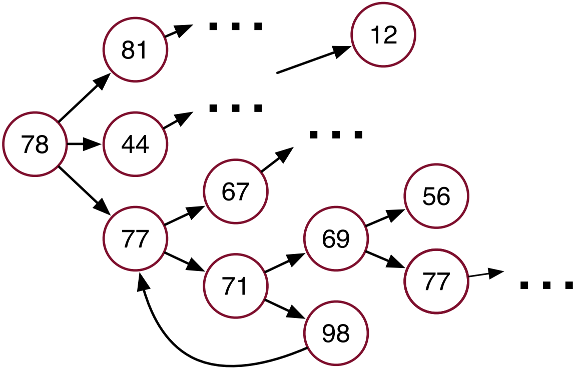
$$\mathcal{C}(\texttt{map}\ Func) = C_{init}^{\texttt{map}} + C_{op}^{\texttt{map}} * \mathcal{C}(Func)$$

Program variant exploration

searches based on the cost (gradient descent)

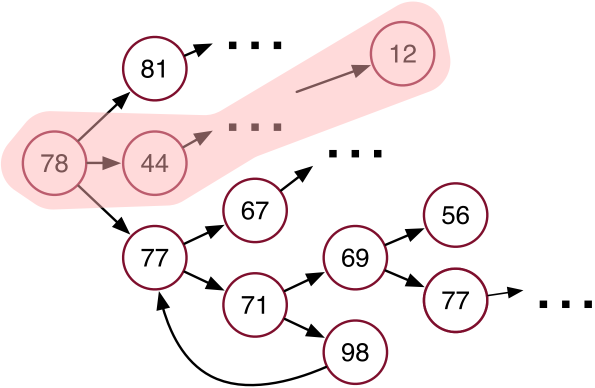


another platform? different cost function



Program variant exploration

different cost function ⇒ different resulting programs



evaluation suite

- applied MOLD on 7 programs (Phoenix benchmark suite[1])
  - WordCount
  - Image Histogram
  - LinearRegression
  - StringMatch
  - MatrixProduct
  - Principal Component Analysis (PCA)
  - K-Means

[1] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis. Evaluating MapReduce for multi-core and multiprocessor systems. HPCA ’07

evaluation methodology

Can MOLD generate *effective* MapReduce code?

Can MOLD generate *effective* MapReduce code?

- check semantics preservation

evaluation methodology

Can MOLD generate *effective* MapReduce code?

- check semantics preservation
- manually inspect the generated code
  - no redundant computation
  - high level of parallelism
  - accesses to large data structures should be localized

Can MOLD generate *effective* MapReduce code?

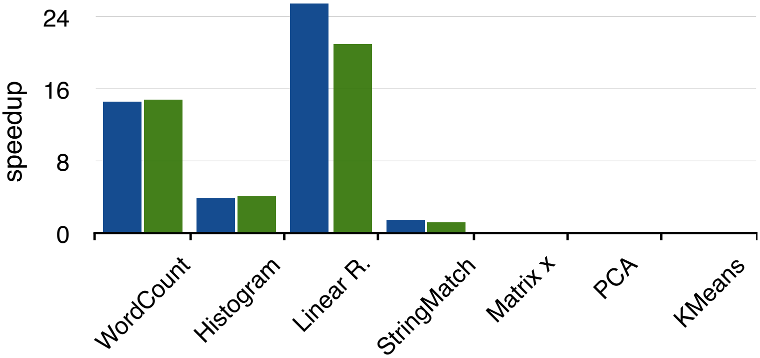
- no redundant computation — 5/7 programs
- parallelism — optimal for 4/7 programs
- memory accesses are localized — 5/7 programs

evaluation methodology

Can MOLD generate *effective* MapReduce code?

- check semantics preservation
- manually inspect the generated code
  - no redundant computation
  - high level of parallelism
  - accesses to large data structures should be localized
- execute with the three backends, compare with hand-written implementations:
  - Scala sequential collections
  - Scala parallel collections
  - Spark

comparison with hand-written implementations



- baseline: hand-written Scala using sequential collections
- blue is hand-written
- green is generated

Conclussions

- we propose an approach for transforming sequential imperative code to functional MapReduce
- sequential imperative $\Rightarrow$ Array SSA $\Rightarrow$ lambda with `fold`
- search through a space of possible optimizations
  - transformations are expressed as rewrite rules
  - generic handling of indirect updates
  - cost function can be platform-dependent
- good results on a small set of benchmarks

Appendix

Cost estimation

$$
\begin{aligned}
\mathcal{C}(F \circ G) &= \mathcal{C}(F) + \mathcal{C}(G) \\
\mathcal{C}(F(G)) &= \mathcal{C}(F) + \mathcal{C}(G) \\
\mathcal{C}(\langle F, G, ... \rangle) &= \mathcal{C}(F) + \mathcal{C}(G) + ... \\
\mathcal{C}(A[I]) &= C^{collection}_{get} + \mathcal{C}(A) + \mathcal{C}(I) \\
\mathcal{C}(A[K \texttt{ := } V]) &= C^{collection}_{set} + \mathcal{C}(A) + \mathcal{C}(K) + \mathcal{C}(V) \\
\mathcal{C}(\texttt{map}\, F) &= C^{\texttt{map}}_{init} + C^{\texttt{map}}_{op} * \mathcal{C}(F) \\
\mathcal{C}(\texttt{fold}\, I\ F) &= \mathcal{C}(I) + C^{\texttt{fold}}_{init} + C^{\texttt{fold}}_{op} * \mathcal{C}(F) \\
\mathcal{C}(\texttt{groupBy}\, F) &= C^{\texttt{groupBy}}_{init} + C^{\texttt{groupBy}}_{op} * \mathcal{C}(F)
\end{aligned}
$$

Transformation rules

(extract map from fold)

$$\frac{\texttt{fold}\langle r_0^0, \ldots, r_n^0 \rangle\ \lambda \langle r_0, \ldots, r_n \rangle\ K\ V\ .\ E}{\begin{array}{c}(\texttt{fold}\langle r_0^0, \ldots, r_n^0 \rangle\ \lambda \langle r_0, \ldots, r_n \rangle\ K\ \langle v_0^f, \ldots, v_m^f \rangle V_{\cap\,\mathrm{free}(F)}\ .\ F) \\ \circ\ (\texttt{map}\ \lambda\ K\ V\ .\langle G[r_\_^0/r_\_], V_{\cap\,\mathrm{free}(F)} \rangle)\end{array}}$$

$E = (\lambda \langle v_0^f, \ldots, v_m^f \rangle\ .\ F) \circ G$

F is $\arg\max \mathcal{C}(G)$ with the condition:

$\nexists i \in [0..n]\ .r_i \in G \wedge r_i \in E[r_\_^0/r_\_]$

where

$r_\_^0/r_\_ = r_i^0[k]/r_i[k]$ applied for all $i \in [1..n]$ $k \in K$

(fold to group by)

$$\frac{\texttt{fold}\, r_0\, \lambda\, r\, V\ .\ r[E := B]}{\begin{array}{r}(\texttt{map}\, \lambda\, k\, l\ .(\texttt{fold}\, r_0[k]\, \lambda\, g\, V\ .\ C)\ l) \\ \circ\ (\texttt{groupBy}\, \lambda\, V\ .\ E)\end{array}}$$

$C = B[g/r[E]]$

$r \notin C \wedge r \notin E \wedge \exists\, v \in V.v \in E$

we cannot prove E is distinct across the folding

Is the proposed approach general?

algorithm	loops/ loop nests	translation time (s)	transformations
WordCount	2/1	11	15
Histogram	1/1	233	18
LinearRegression	1/1	28	2
StringMatch	1/1	68	2
Matrix $\times$	3/1	40	20
PCA	5/2	66	15
KMeans	6/2	340	10