GTA*

An Algebraic Method for Developing Divide and Conquer Algorithms

Sebastian Fischer
(with Kento Emoto and Zhenjiang Hu)

*Generate, Test, and Aggregate
Divide

Problem

... ...

Conquer

Sub-Problem

Sub-Solution

Solution

Sub-Problem

Sub-Solution

Combine
Sorting

[3, 1, 5, 2, 6, 4, 7]

Split

[3, 1, 5]

Sort

[1, 3, 5]

Sort

[2, 6, 4, 7]

Merge

[2, 4, 6, 7]

[1, 2, 3, 4, 5, 6, 7]
Knapsack Problem

- Fill knapsack with items
- Each has value and weight
- Problem:
  
  maximise total value
  without exceeding weight restriction
Divide and Conquer?

- $200¥$ 1 kg
- $100¥$ 3 kg
- $400¥$ 3 kg
- $5$ kg

max $5$ kg
Divide and Conquer

- Important design pattern
- Useful for parallel programming
  \( \rightarrow \) MapReduce
- Can be (and often is!) tricky in practice
GTA

- hides "trickiness"

- general method applicable to wide class of problems

- automatic parallelization
G: simple divide and conquer algorithm to generate intermediate results

T: simple divide and conquer algorithm to test each intermediate result

A: "kind of" divide and conquer algorithm to aggregate remaining results
GTA

- makes complex divide and conquer algorithm from simpler ones
- specification as a search problem
- possibly many intermediate results
- turns inefficient search into efficient parallel program
Knapsack Problem

G: generate (multi set) of all possible combinations of items
   easy 🔺

T: discard all combinations with a too big total weight
   easy 🔺

A: among remaining combinations select one with maximum value
   easy 🔺
Knapsack Problem

Intuition:
- too many intermediate results
  \(2^n\) if \(n\) is number of items

Nevertheless:
- efficient algorithm can be obtained automatically
How ???
Using Math !!!
Some Algebra

- Monoids, Lists
- Semirings, Multisets of Lists
- Homomorphisms
Monoids

set \( M \) with associative binary operation \( \otimes \) and identity element \( id \).

example: lists with concatenation

\[(x +\otimes y) +\otimes z = x +\otimes (y +\otimes z)\]

\[\text{[]} +\otimes x = x = x +\otimes \text{[]}\]
List Homomorphism

\[ h[\ ] = id \circ \emptyset \]
\[ h[x] = f \circ x \]
\[ h(x+y) = h(x) \circ h(y) \]

\( \rightarrow \) divide and conquer on lists
Semirings

[Diagram showing two monoids $(\mathcal{M}, \otimes)$, $(\mathcal{M}, \Theta)$ with distributivity and commutativity properties.]

- Two monoids $(\mathcal{M}, \otimes)$, $(\mathcal{M}, \Theta)$
- Distributivity:
  \[ x \otimes (y \Theta z) = (x \otimes y) \Theta (x \otimes z) \]
  \[ (x \otimes y) \Theta z = (x \otimes z) \Theta (y \otimes z) \]
- Commutative laws:
  \[ \text{id} \otimes x = \text{id} \Theta \]
  \[ x \Theta \text{id} \Theta = \text{id} \Theta \]
Example

(\mathbb{Z} \cup \{-\infty\}, \max)

id_+ = 0

id_{\max} = -\infty

x + \max y z = \max (x+y) (x+z)

-\infty + x = -\infty
Another Example

\[ (\llbracket a \rrbracket_S, X_{++}) \oplus \) \\

multisets (bags) of lists

bag union

\[ a X_{++} b = \{ x_{++} y \mid x \in a, y \in b \} \]
Bags of Lists

\([\{1\}\{S \cup \{1,2\}\} \times \tt \{3\}S = \]
\([\{1\}, \{2\}\{S \times \tt \{3\}S = \]
\([\{1\} \tt \{3\}, \{2\} \tt \{3\}\}S = \]
\([\{1,3\}, \{2,3\}\}S\]
Semiring Homomorphisms

\[ h \mathcal{S} = \text{id} \oplus \]

\[ h \mathcal{L} = \text{id} \otimes \]

\[ h \mathcal{[x]} S = f \times \]

\[ h(a \oplus b) = h a \oplus h b \]

\[ h(a \otimes b) = h a \otimes h b \]
if

\[ G: \text{list homomorphism generate} \]

\[ T: (\text{almost}) \text{ list homomorphism test} \]

\[ A: \text{semiring homomorphism aggregate} \]

then

aggregate . filter test . generate

\[ (\text{efficient}) \text{ divide \& conquer alg.} \]
Ok, but how ??
Secret!

(but see paper)
Knapsack Generator

\[
\begin{align*}
\text{sublists } [] &= 2\times[5] \xrightarrow{\text{id}} x_{++} \\
\text{sublists } [\text{item}] &= 2\times[5] \oplus 2\times[\text{item}] \\
\text{sublists } (a+b) &= \text{sublists } a \times_{++} \text{sublists } b \\
\rightarrow \text{ list homomorphsim}
\end{align*}
\]
Knapsack Test

weight [] = 0
weight [(r,ω)] = ω
weight (a + b) = weight a + weight b

test items = weight items ≤ W

→ almost
  list homomorphism

→ maximum weight
Knapsack aggregator

(not essential) simplification:
compute maximum possible value
rather than corresponding items

\[ \text{maxval} \quad |S| = 0 \]
\[ \text{maxval} \quad |\{1\}|S = 0 \]
\[ \text{maxval} \quad |\{(0,\omega)\}|S = \omega \]
\[ \text{maxval} \quad (a \& b) = \max (\text{maxval} a, \text{maxval} b) \]
\[ \text{maxval} \quad (a \lor b) = \text{maxval} a + \text{maxval} b \]

→ semiring homomorphism
Efficient Knapsack Algorithm

\[
[(200¥, 1kg), (300¥, 3kg), (400¥, 3kg)]
\]

Divide

- 1kg: 200¥
- 3kg: 300¥
- 3kg: 400¥

Combine

3kg: 300¥
3kg: 400¥

\rightarrow \text{max value: 600¥}
Associativity

\[ (\text{200￥, 1kg}), (\text{300￥, 3kg}), (\text{400￥, 3kg}) \]

Divide

1kg : 200￥
3kg : 300￥
4kg : 500￥

Combine

3kg : 300￥

Combine

3kg : 400￥

Combine

7kg : 200￥
3kg : 400￥
4kg : 600￥

28.8
Knapsack Complexity

- linear in number of items
- quadratic in maximum weight
  \( \rightarrow \) pseudo polynomial
- one processor: \( O(nw^2) \)
- \( p \) processors: \( O((\log p + \frac{n}{p})w^2) \)
GTA

efficient divide and conquer algorithm
from intuitive specification if

G: list homomorphism
T: (almost) list homomorphism
A: semiring homomorphism

knapsack example generalizes
to other applications, complexity too
many predefined generators:
sublists, prefixes, suffixes, segments, ...

practical applications:
- inferring states of hidden Markov model
- incremental refinement via filters

generalizes to:
- other input types easy
- other intermediate types possible too
Secrets Revealed

Emoto, F. 11th

Generate, Test, and Aggregate —
A Calculation-based Framework for Systematic Parallel Programming with MapReduce

ESOP 2012
ありがとう！