

プログラム演算に基づく最適化機能つきFortressライブラリ

江本 健斗¹、胡 振江²、寛 一彦³、松崎 公紀¹、武市 正人¹

¹ 東京大学 大学院情報理工学系研究科、² 国立情報学研究所 アーキテクチャ科学研究所、³ 東京大学 産学連携本部

Sun Microsystems との共同研究

目的: プログラム構築の技術を新しい言語Fortressへ

• 本共同研究は、我々の研究してきたプログラム構築のための技術である、並列スケルトン、プログラム演算、並列性の自動導出などを、Fortress上の容易かつ頑健な並列プログラミングの実現のために応用することを目的とする。新しい並列プログラミング言語Fortress は、様々な関数プログラミングの特徴を持ち、我々の関数型プログラミングを基盤とした技術と相性が良いと考えられる。

研究の第一歩: ネストした生成と縮約の計算を対象に

• 要素の集まりの生成とその上の縮約は、並列計算の基本パターンであり、FortressではGeneratorがそれを抽象化する。多くの応用問題では、この計算をネストさせ、要素の集まりの集まりを生成してその上での多段の縮約を行う。しかし、プレフィックスの集まりのように、生成された集まりの間に依存関係がある場合には、Generatorの提供する単純な計算は非効率になり得る。そこで、我々は、依存のある集まりの生成を抽象化し、プログラム演算を応用した最適化の実現を目指す。

成果: プログラム演算を応用した最適化ライブラリ

• プログラム演算を利用した最適化をライクウェアに実現。その効果は非明かつ大胆。
• 現在、Fortressの標準ライブラリの一部として組み込まれている。将来の仕様へ反映予定。
参考URL: <http://projectfortress.sun.com/Projects/Community>, <http://www.ipl.t.u-tokyo.ac.jp/fortress/>

ネストした生成・縮約の抽象化と最適化の仕組

既存のGeneratorプロジェクト

- 生成と縮約の並列計算を抽象化
 - 生成関数と縮約演算子を取るメソッドgenerateを持つ
- 例: 1, 2, 3, 4の自乗和

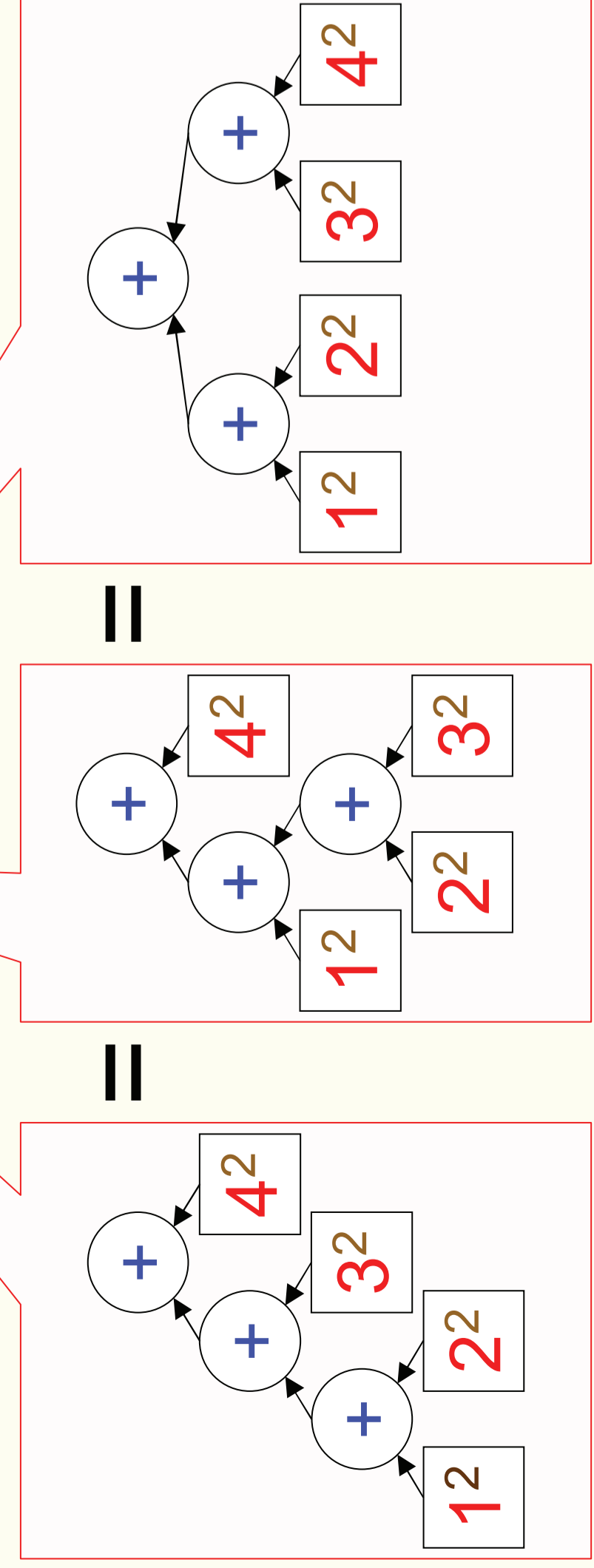
```
SUM [ x2 | x <- (1#4) ]
```



Desugaring

```
(1#4).generate(SUM, fn x => x2)
```

- 結合性を用い計算構造を柔軟に変化



新しいGoG (Generators of Generators) オブジェクト

- ネストした生成と縮約の計算を抽象化
- 生成関数と2つの縮約演算子を取るメソッドgenerate2
- 演算子を2つ取ることで定理の適用条件判定が可能

```
OP [ OT [ f x | x <- g ] | g <- gg ]
```



Desugaring

```
gg.generate2(OP, OT, fn x => f x)
```

- 適用可能な定理の実装を選択し最適化

定理1 (gg = inits) 条件

```
OTがOIに分配
実装
OD[(a, a)|a<-x]
opr OD ... =
do
  ...OT...OP...
end
```

• 演算子オブジェクトに型で情報を付加

```
trait Prod extends Distr[¥Sum¥] ...end
trait Distr[¥E¥] end
```

• 型を見ることで演算子間の性質を確認

```
distr[¥Q, R¥](q:Q, r:R) =
  typecase (q, r) of
    (Q, Distr[¥Q¥]) => true
  else => false
end
```

ライブラリの効果 (例: 最大昇部分列和問題)

☺ After

①よく使う生成パターンを抽象化

- 添え字不要の簡潔な記述
- 生成パターンを容易に切替可

```
Naive program O(n3)
BIG MAX [ SUM y | y <- segs x,
           ascending y ]
```

☹ Before

添え字の複雑な使用の問題

- 間違った元の
- 見難く意味が読めない

```
Naive program O(n3)
BIG MAX [ SUM x[b#s] | b <- 0#|x|,
           s <- 1#(|x|-b),
           ascending(x[b#s])] ]
```

○ × 定理 (運算規則)

演算子 ⊗ が演算子 ⊕ に分配し、述語 p がある形に定義できるなら、
⊕ / ⊙ (⊗ / *) ⊙ p ⊙ segs = fst ⊙ ⊙ (⊗, ⊕) / ⊙ tup*

Efficient program O(n)

```
fst( BIG OD [ tup y | y <- x ] )
opr OD(a, b) =
do (a1, a2, a3, a4, a5, a6) = a;
```

②定理をライブラリが自動適用

- 定理の適用を誤らない
- 複雑な実装を書く必要なし

定理の手動での適用の問題

- 条件を満たさぬ定理の誤用
- 結果の実装が複雑で間違っ



自分で最適化するか?

実験結果

(Fortress interpreter (r3294) on a PC with two quadcore CPUs (Intel® Xeon® E5430, total 8 cores), 8GB memory, Linux 2.6.24.)

MIS: BIG MAX [SUM y | y <- inits x]
最大先頭部分列和

MTP: BIG MIN [PROD y | y <- tails x]
最小末尾部分列積

MSS: BIG MAX [SUM y | y <- segs x]
最大部分列和

pMIS: BIG MAX [SUM y | y <- inits x, ascending y]
要素が昇順に並んだ先頭部分列の最大和

pMTP: BIG MIN [PROD y | y <- tails x, descending y]
要素が降順に並んだ末尾部分列の最小積

pMSS: BIG MAX [SUM y | y <- segs x, flat4 y, ascending y]
要素が昇順に並びかつ隣同士の差が4未満の部分列の和の最大値

非自明な最適化で線形コストに