Using Algebraic Transformations to Optimize Expression Evaluation in Scientific Codes

Julien Zory and Fabien Coelho

—

Centre de Recherche en Informatique

École des Mines de Paris

http://www.cri.ensmp.fr/pips/EOLE

October 1998

- **Problem**

  Repeated evaluation of quite large expressions

  Improve expression evaluation for superscalar architectures

- **Approach**

  Explore the mathematically equivalent expression space

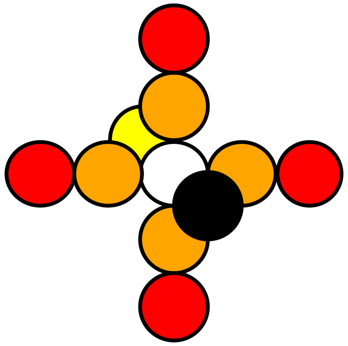
  Study the impact of algebraic transformations on performance

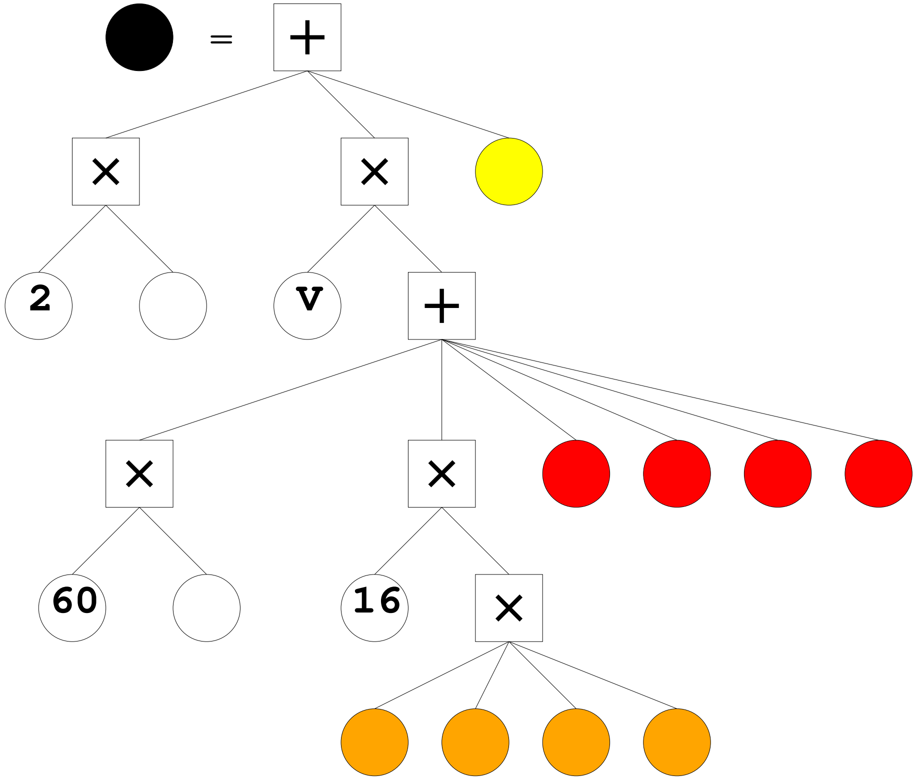
- **Constraints**

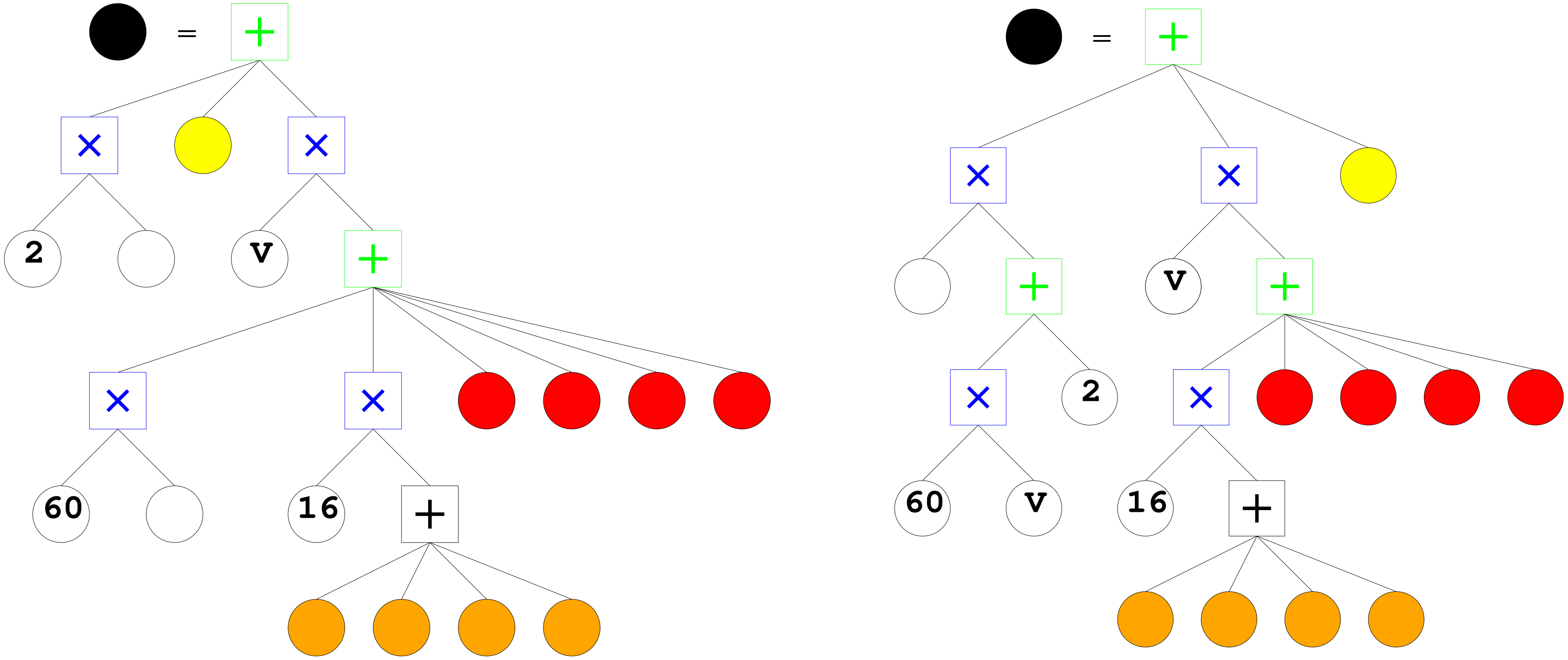
  Avoid combinatorial explosion

  Take into account ILP and new instructions

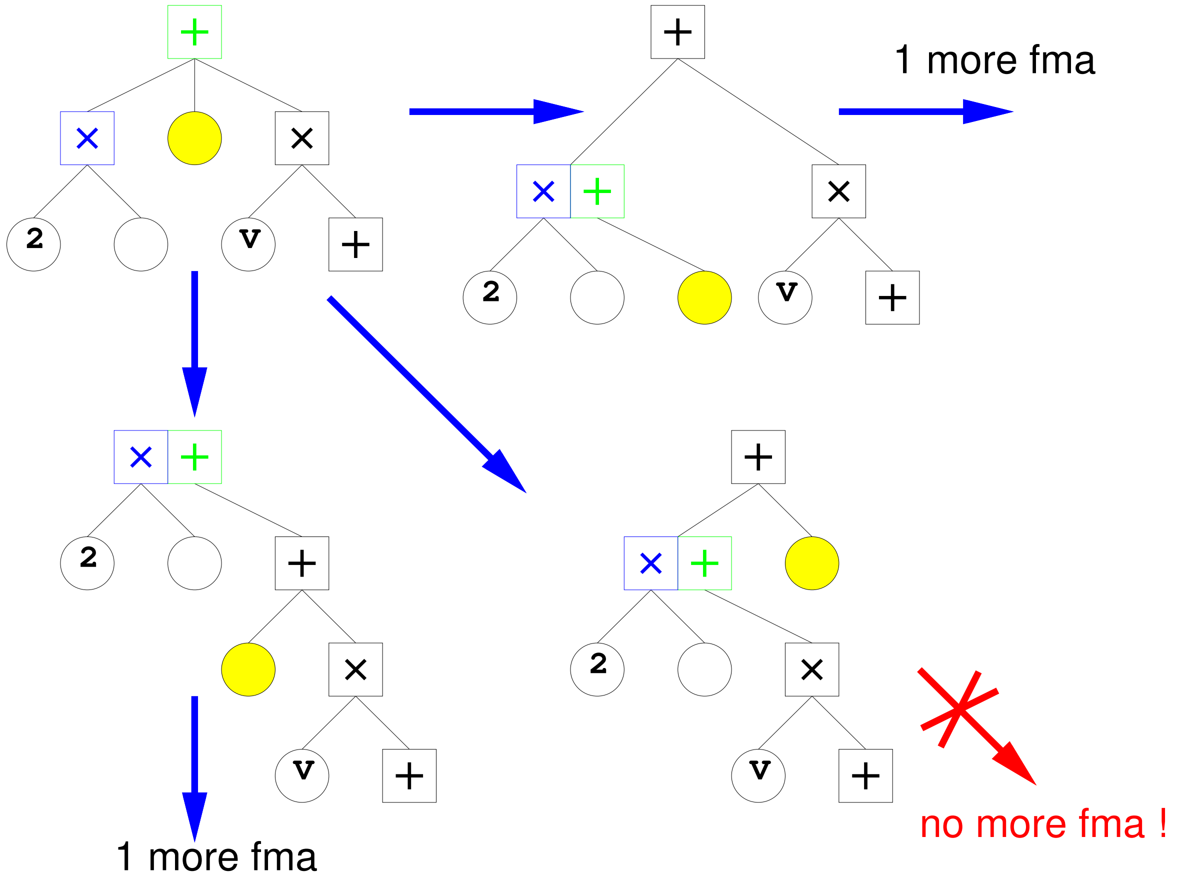
```
[...]
 DO J = 3,NP-2
   DO I = 3, NP-2
     U(I,J,KP) =
$      2 * U(I,J,KM) - U(I,J,KP)
$      - V(I,J) * ( 60*U(I,J,KM)
$      -16*( U(I+1,J,KM) + U(I-1,J,KM)
$      + U(I,J-1,KM) + U(I,J+1,KM))
$      + U(I+2,J,KM) + U(I-2,J,KM)
$      + U(I,J-2,KM) + U(I,J+2,KM))
   ENDDO
 ENDDO
[...]
```







2 factorizations and 4 multiply-adds



69300 mathematically equivalent expressions

- **Architecture**

  120 & 160 MHz IBM RS6000 processors (P2SC 595)

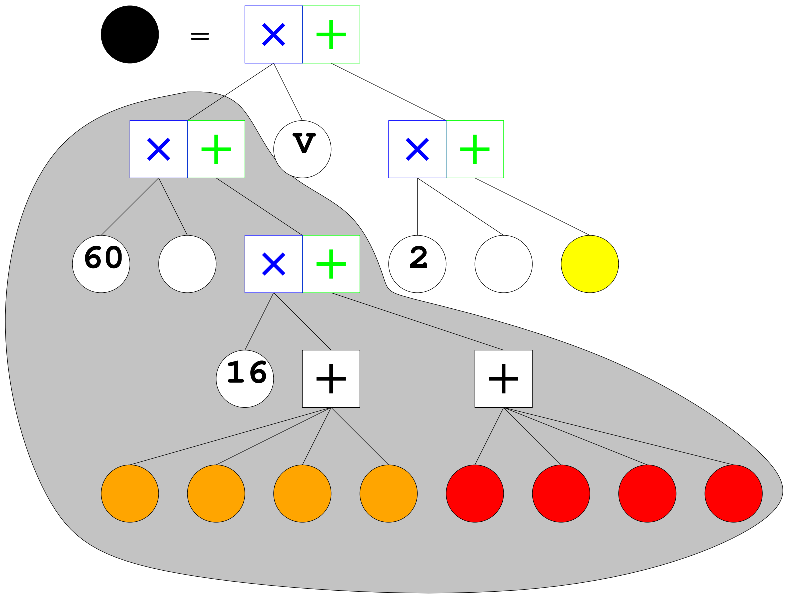
  2 floating-point units
  2 stage floating-point pipeline
  } 4 independent operations

  multiply-add instruction (same latency as addition)

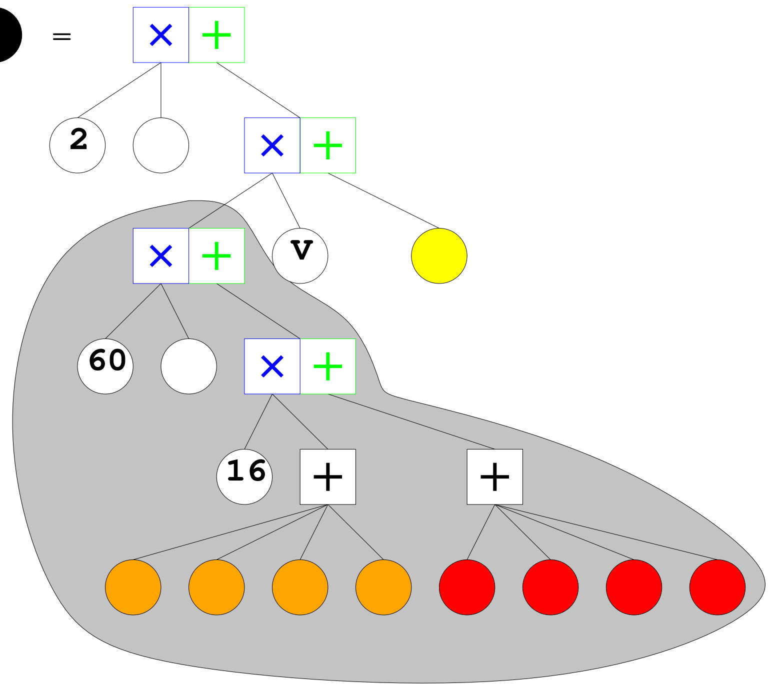
- **Compilation**

  IBM XLFortran release 4.1 (-O3 -qarch=pwr2 -qhot)

  KAP preprocessor for IBM Fortran (V3.3)



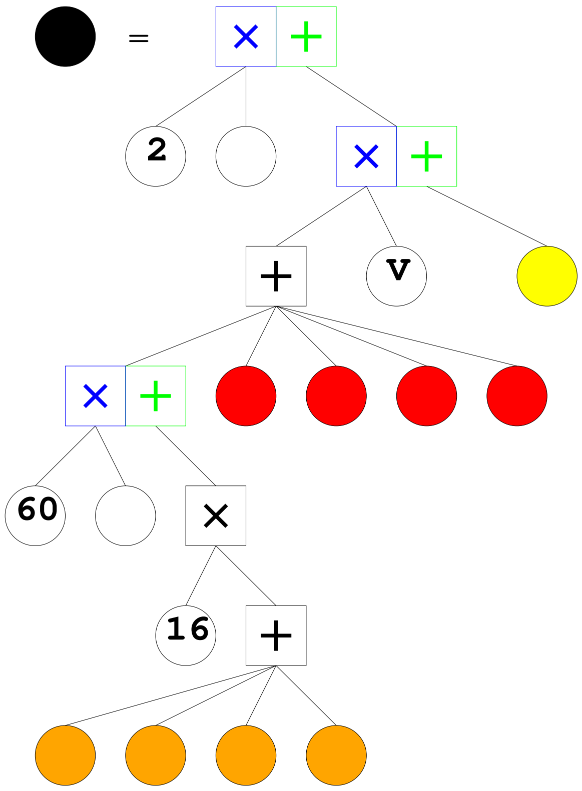
149 MFLOP/s



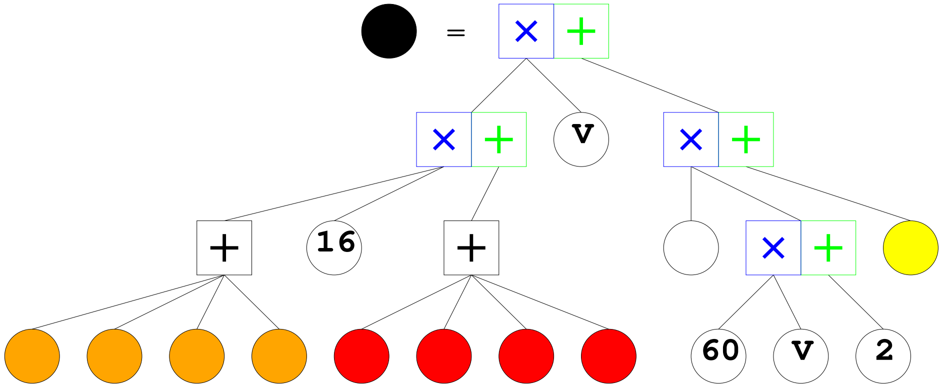
130 MFLOP/s

➡ 2 different critical paths

ONDE24 - Some results on a 160MHz P2SC 595



102 MFLOP/s



162 MFLOP/s

➡ A 60% performance improvement

➢ Our approach

Search the space of algebraic equivalent expressions

Reach a *good* final expression

➢ However

The expression space is **too large**

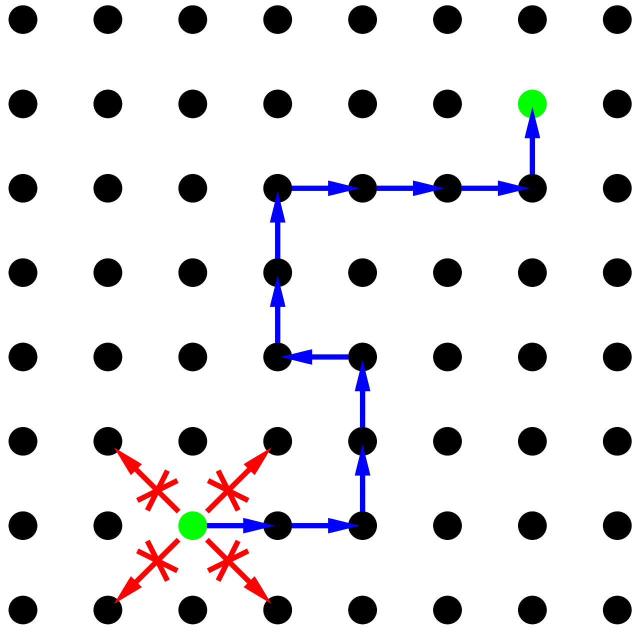
➢ Thus ... Greedy heuristics

Follow an **iterative** process

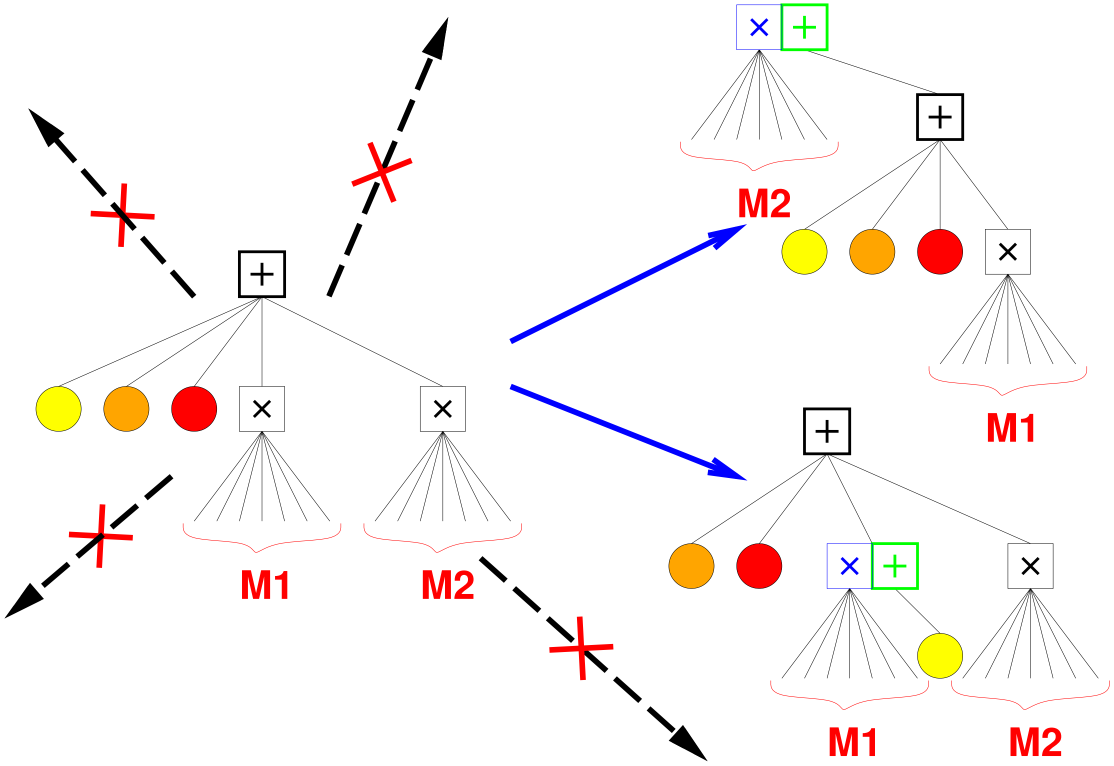
Take **local decisions**

Limit the space of **elementary transformations**

Find out simple and efficient **choice criteria**



2 interesting **elementary transformations** selected



Choice criteria

➡ ~~real cost~~

➡ estimated cost

- $W = \sum_{i=1}^{n} w_i$
- $C = \max_{i=1}^{n} d_i$
- $G = \frac{\sum_{i=1}^{n} w_i \times d_i}{W}$

Greedy heuristics :

select a transformation according to the choice criterion

A 3 step process

1. Normalization - *pattern matching*
   - ensure that expressions can be freely rearranged
   - apply easy algebraic simplifications
2. Factorization - *greedy heuristics*
   - reduce the number of operations
   - choose the most balanced tree
   - favor later invariant code motion
3. Multiply-add extraction - *greedy heuristics*
   - extract as many multiply-adds as possible
   - choose the most balanced tree

- Algebraic transformation contributions

  reduce the **number of operations**

  benefit from **new instructions** (e.g. fma)

  enhance **ILP** for the compiler

- Approach

  3 step process

  greedy heuristics

  simple and efficient **choice criteria**

- Very promising experimental results
  up to factor 3 performance improvements

- Fully implemented - EOLE project
  use CAVEAT (CEA) and STORM (LIFO,Stony Brook)
  included in the PIPS tool (Source to Source Fortran)

- Reasonable optimization time (compilation context)
  ONDE24 (500 lines) ➡ 4 sec (143 MHz Ultra1)

Future Work

- More experiments
  - SPEC FP95 *(see the paper)*
  - Perfect Club
  - Automatically generated code (HPF)
- Wider range of processors
  - IBM Power2 and P2SC
  - MIPS R10000
  - *HP PA-RISC*
  - *INTEL Merced*
- Associative-commutative framework extension for
  - Invariant code motion
  - Common sub-expression elimination

−

- floating-point arithmetic well-known instability
- language specifications

  Fortran - must respect parentheses

  C - firstly allowed but later rejected transformations

- has it been written with numerical precision in mind ?

- reduce the number of operations
- more precise multiply-add operations

+

➥ high-level aggressive optimizations already in compilers

- Very difficult for the programmer

  require a good knowledge of the low-level capabilities

  legibility and portability losses

- Far from the usual compiler optimizations

  3-address code internal representation

- Easy to re-target