Symbolic Manipulation System FORM: outlines and outlooks

Mikhail Tentyukov

TTP Karlsruhe

CALC-2009, 10-20 July, Dubna.

What is FORM ?

FORM

is a program by J. Vermaseren for symbolic manipulation of algebraic expressions specialized to handle very large expressions of millions of terms in an efficient and reliable way.

http://www.nikhef.nl/ ~ form

publications:

http://www.nikhef.nl/ ~ form/maindir/publications/publications.html

Manual & Tutorials:

http://www.nikhef.nl/ ~ form/maindir/documentation/documentation.html

FORM vs. CAS

From Geert Jan van Oldenborgh, *An Introduction to FORM:*

Maple, Mathematica, etc.

Swiss army knife

- Much built-in knowledge (integration, solving equation, special funclions, etc.)
- Designed for an infinitely large computer
- Big and slow (especially on large problem)
- Very general
- Fancy user integface (typesetting, graphics, sound,drag-and-drop, etc.)
- Tries to do everithing

FORM

Chef knife

- Limited mathematical knowledge (calculus with tensors and gamma matrices, ...)
- Designed for real computers
- Small and fast (also on large problems)
- Optimized for certain classes of problems
- Batch program (edit-run cycle)
- Does only what you ask for

"Hellow World" in FORM

"Hello World" for symbolic manipulation:

```
Symbols a,b;
Local f = (a+b)^2;
Print;
.end
```

File "hello.frm".

Call FORM:

```
form3.3 hello
```

"Hellow World" in FORM

form3.3 hello produces:

```
FORM by J.Vermaseren,version 3.3(Mar 28 2009) Run at: Wed Jun 5
18:00:54 2009
    Symbols a,b;
    Local f = (a+b)^2;
    Print;
    .end

Time =       0.00 sec    Generated terms =          3
               f         Terms in output =          3
                         Bytes used      =        108

   f =
      b^2 + 2*a*b + a^2;

  0.00 sec out of 0.00 sec
```

Substitutions and flow control I

```
Symbols b,a,x;
Local expr = a*x +x^2;
Identify x^2=a+b;
print;
.end
```

Substitutions and flow control I

```
Symbols b,a,c,x;
Local expr = a*x +x^2;
Identify c?^2=a+b;
print;
.end
```

```
Time =        0.00 sec    Generated terms =
              expr        Terms in output =
                          Bytes used      =

   expr =
      b + a*x + a;
```

Substitutions and flow control II

```
Off statistics;
Symbols b,a,x;
l expr = a*x +x^2;
id x=a+b;
print;
.sort

if( count(b,1) == 1);
multiply 4*a/b;
endif;
print;
.end
```

Substitutions and flow control II

```
FORM by J.Vermaseren,version 3.3(Mar 28 2009) Run at: Wed ...
    off statistics;
    Symbols b,a,x;
    Local expr = a*x +x^2;
    id x=a+b;
    print;
    .sort
   expr =
      2*a^2 + 3*b*a + b^2;
    if( count(b,1) == 1);
    multiply 4*a/b;
    endif;
    print;
    .end
   expr =
      14*a^2 + b^2;
  0.00 sec out of 0.00 sec
```

FORM internals

Module by module. Each module:

1. Compilation. 2. Generating. 3. Sorting:

Very long expressions!

Non-interactive interpreter.

User provides a program, Interpreter runs the program.

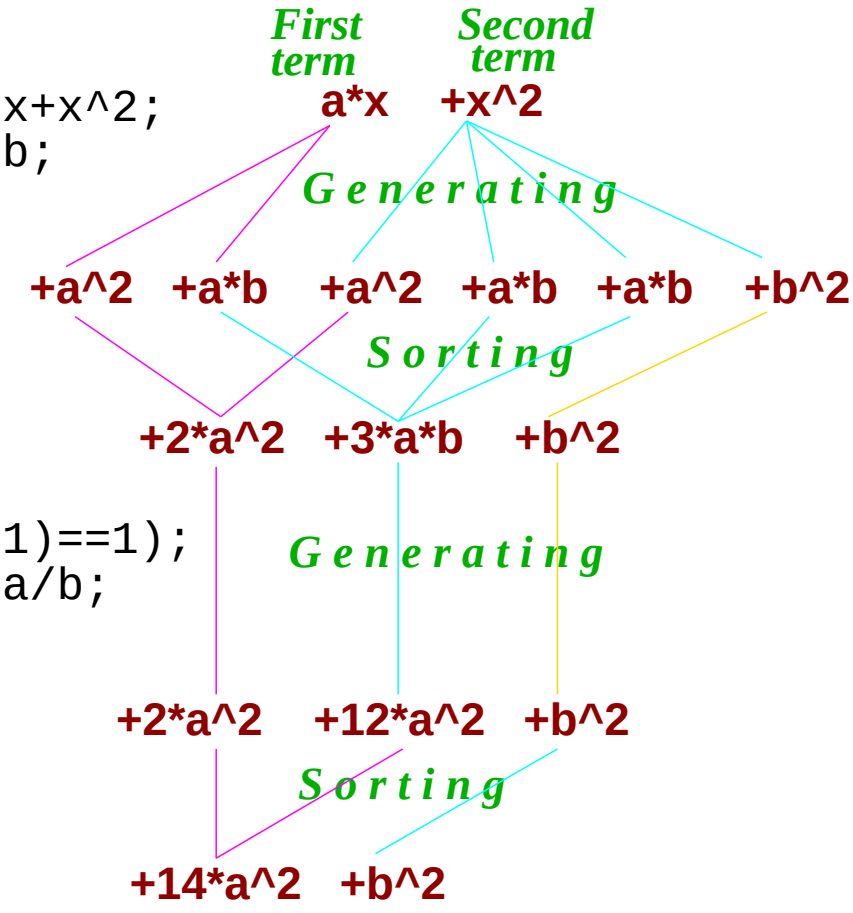
Modules are terminated by “dot” instructions.

First module

```
l expr = a*x+x^2;
id x = a + b;
.sort
```

Second module

```
if(count(b,1)==1);
multiply 4*a/b;
endif;
print;
.end
```



FORM internals

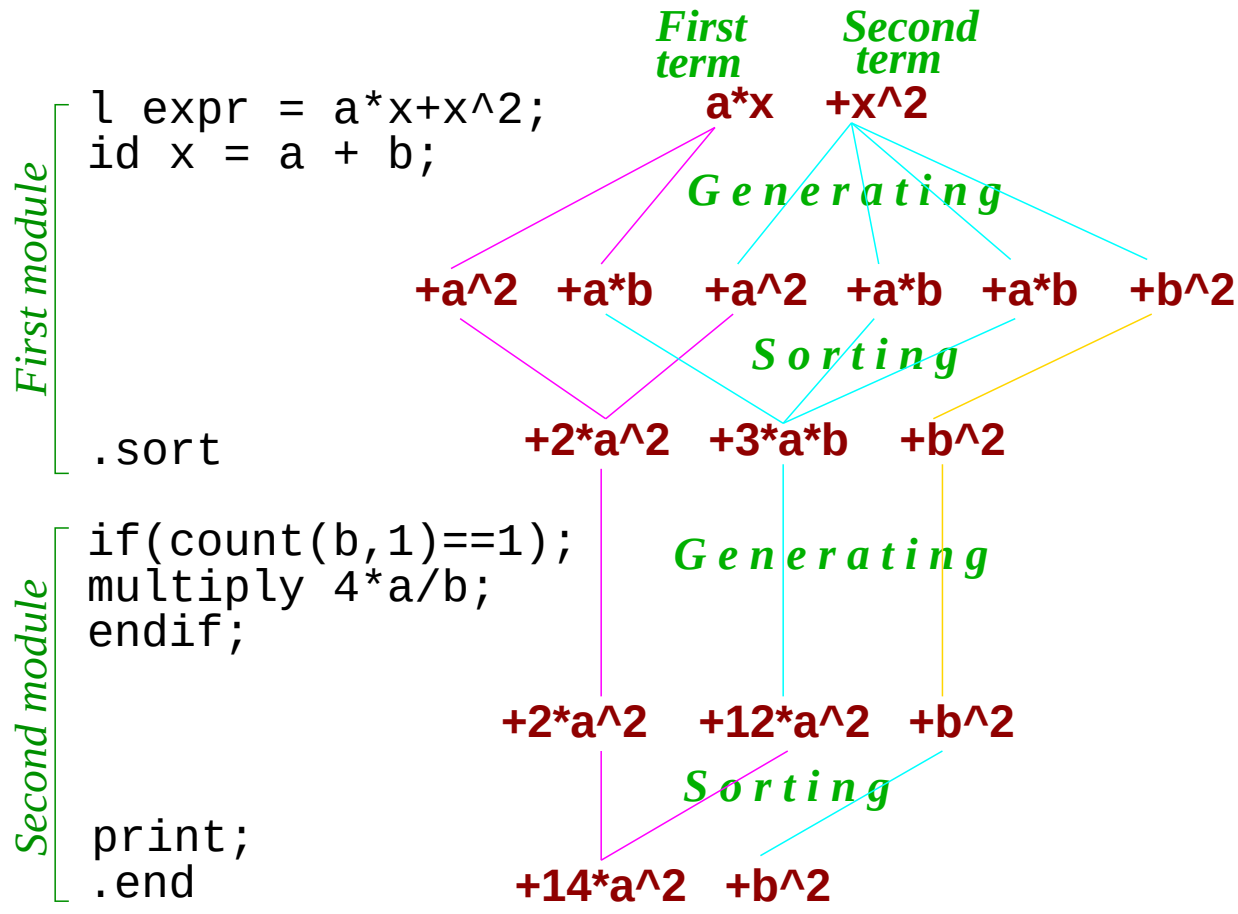
Module by module. Each module:

1. Compilation. 2. Generating. 3. Sorting:

Each module:

Definition of new expressions, algebraic instructions, output instructions.

Expressions remain active through many modules.



FORM internals

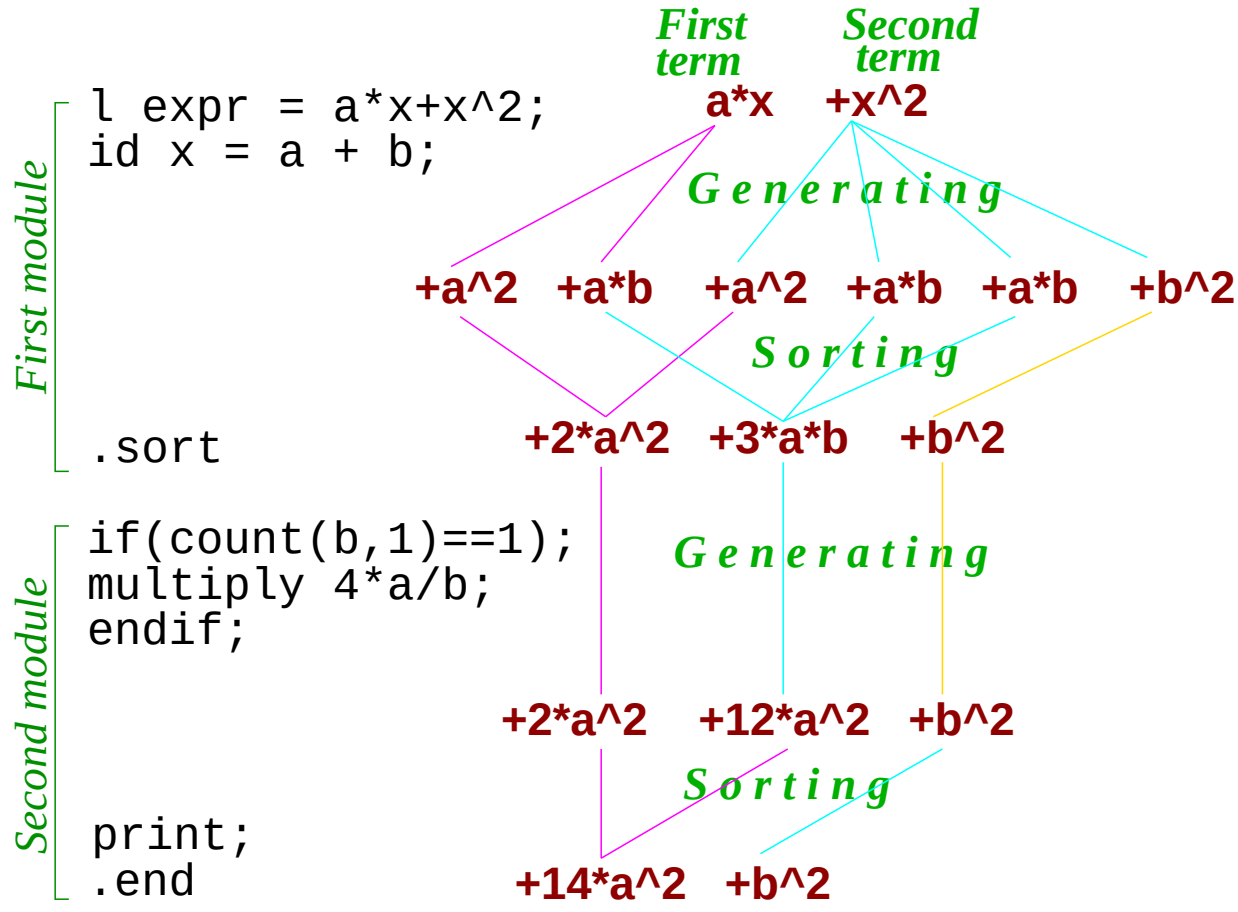
Module by module. Each module:

1. Compilation. 2. Generating. 3. Sorting:

Compilation: Input translated into internal representation.

Generating: Algebraic instructions executed for each term.

Sorting: Generated terms sorted, equivalent terms are summed up.



FORM internals

Module by module. Each module:

1. Compilation. 2. Generating. 3. Sorting:

specific feature:

only *local* operations on single term:

id x = a + b;

Non-local operations are not allowed:

~~id a + b = x~~;

First module

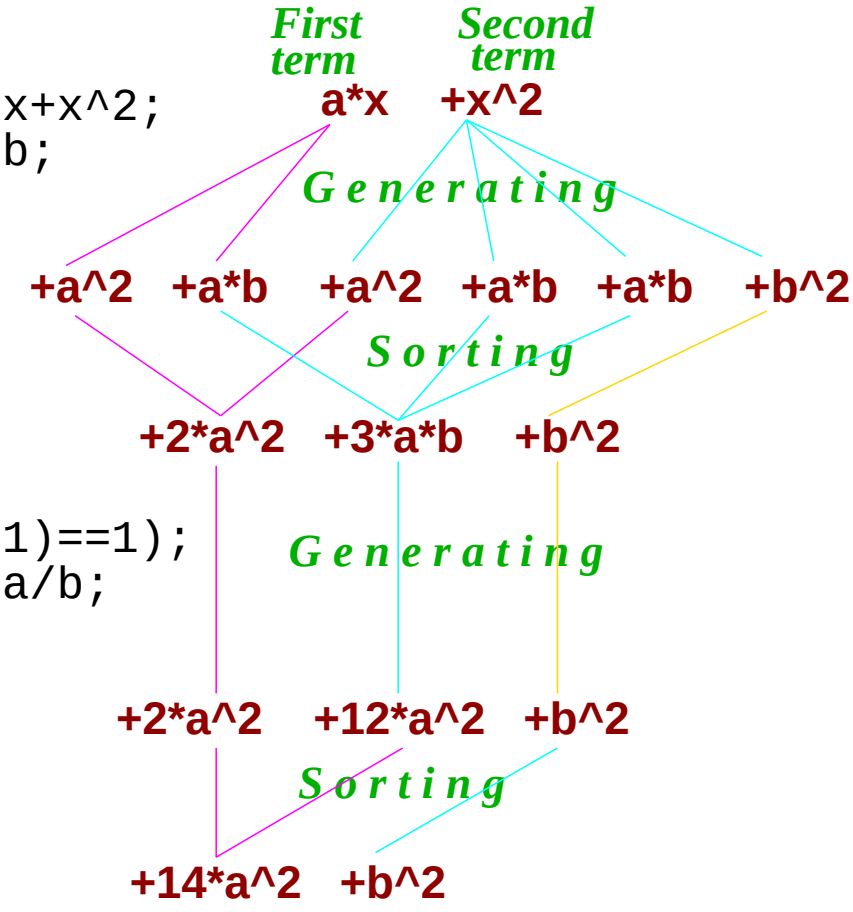
```
l expr = a*x+x^2;
id x = a + b;

.sort
```

Second module

```
if(count(b,1)==1);
multiply 4*a/b;
endif;

print;
.end
```

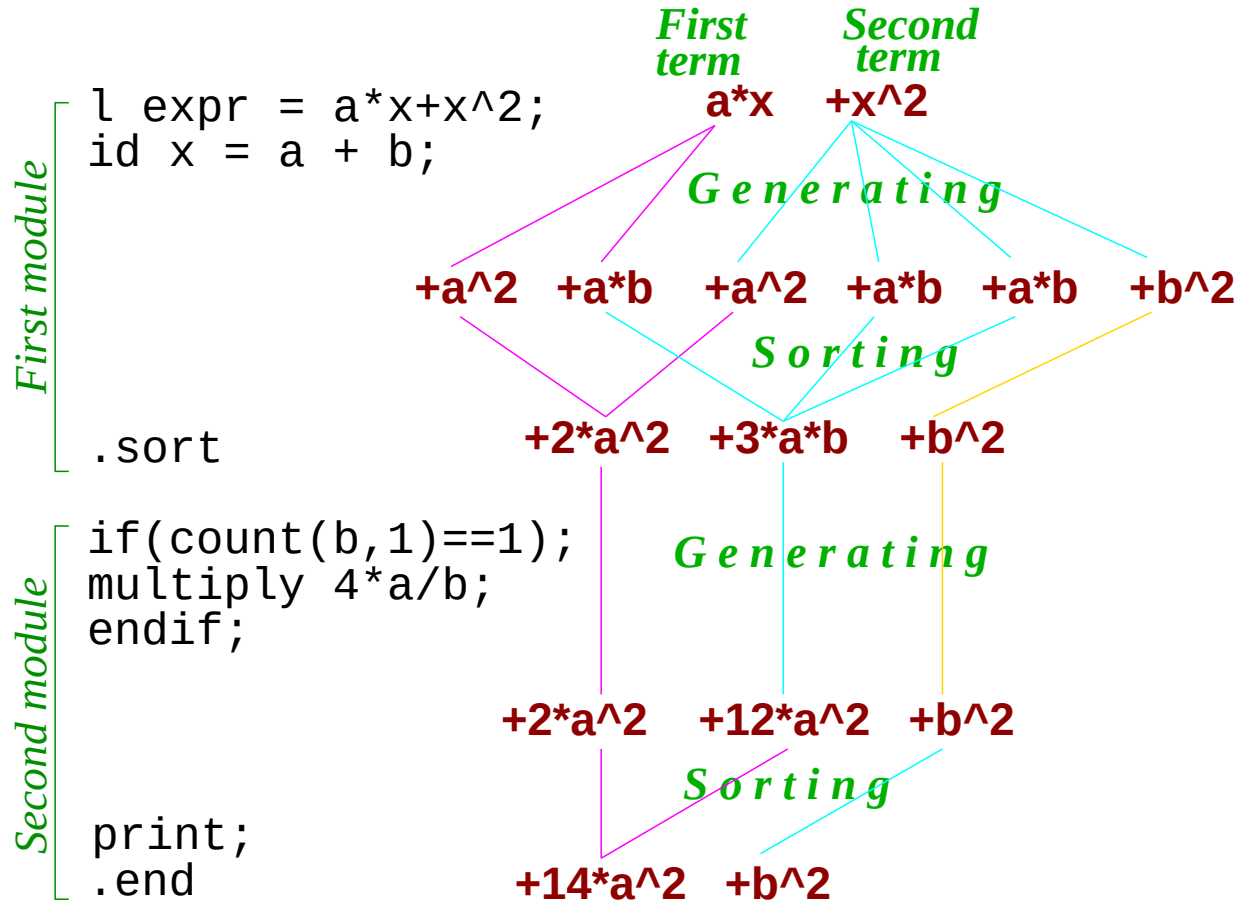


FORM internals

Module by module. Each module:

1. Compilation. 2. Generating. 3. Sorting:

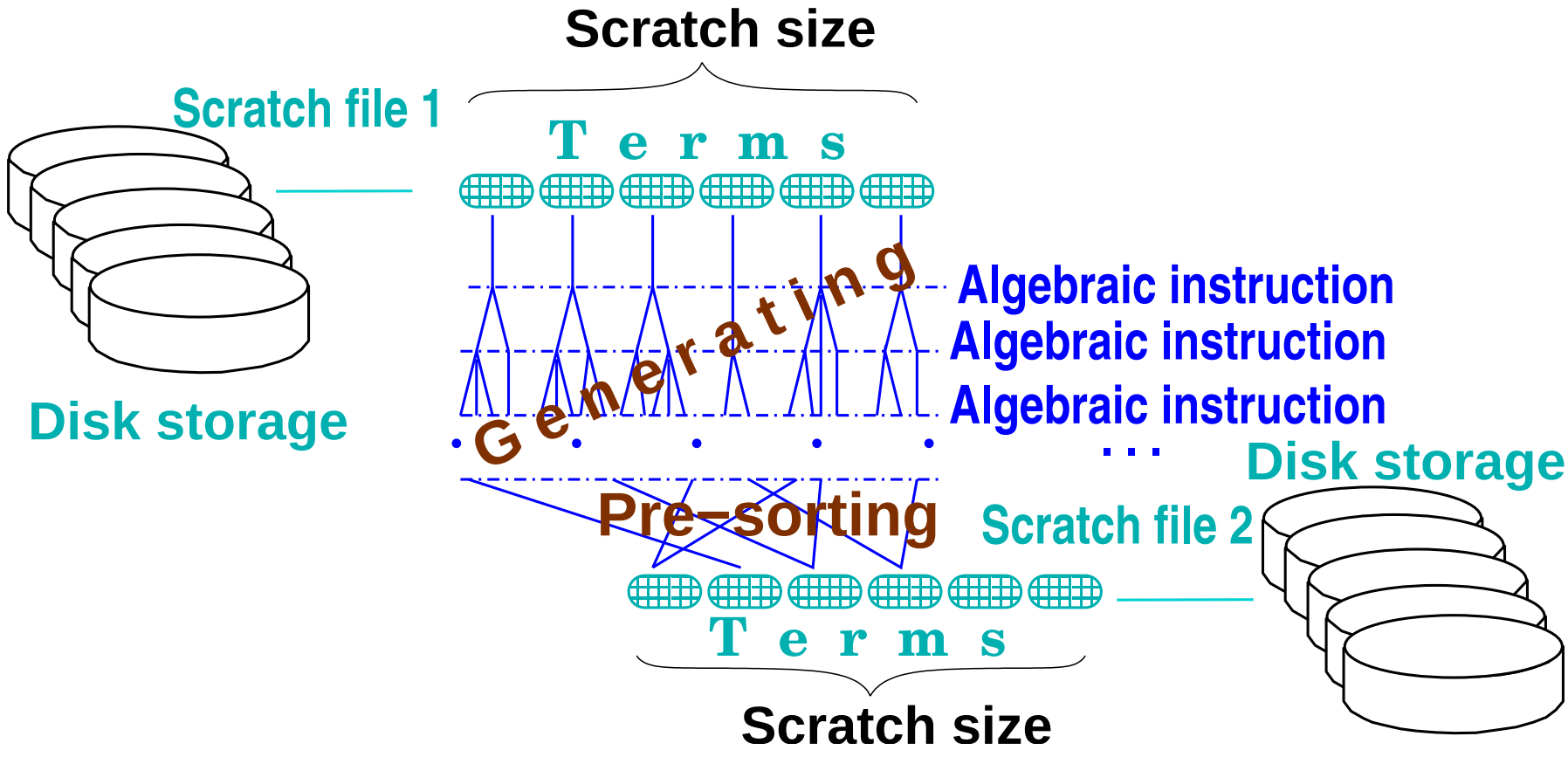
Locality Principle: All *explicit* algebraic operations are local. Non-local operations are allowed only *implicitly* in the sorting procedure at the end of the modules and in some other special cases.



Main FORM feature

Locality Principle → Expressions as “streams” of terms

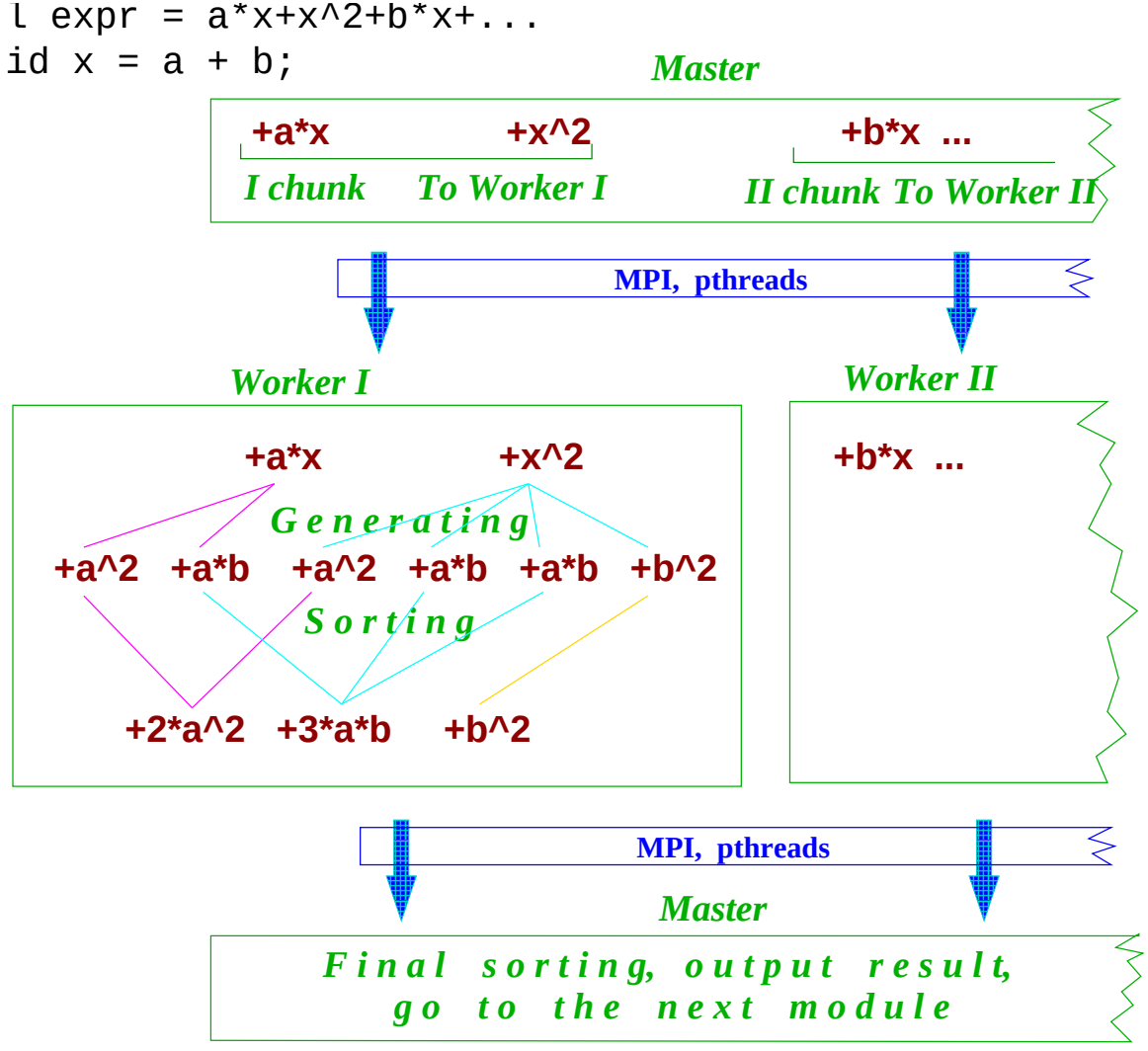
Expressions bigger than the memory (RAM) available!



The concept of FORM parallelization

The *Locality Principle*, again!

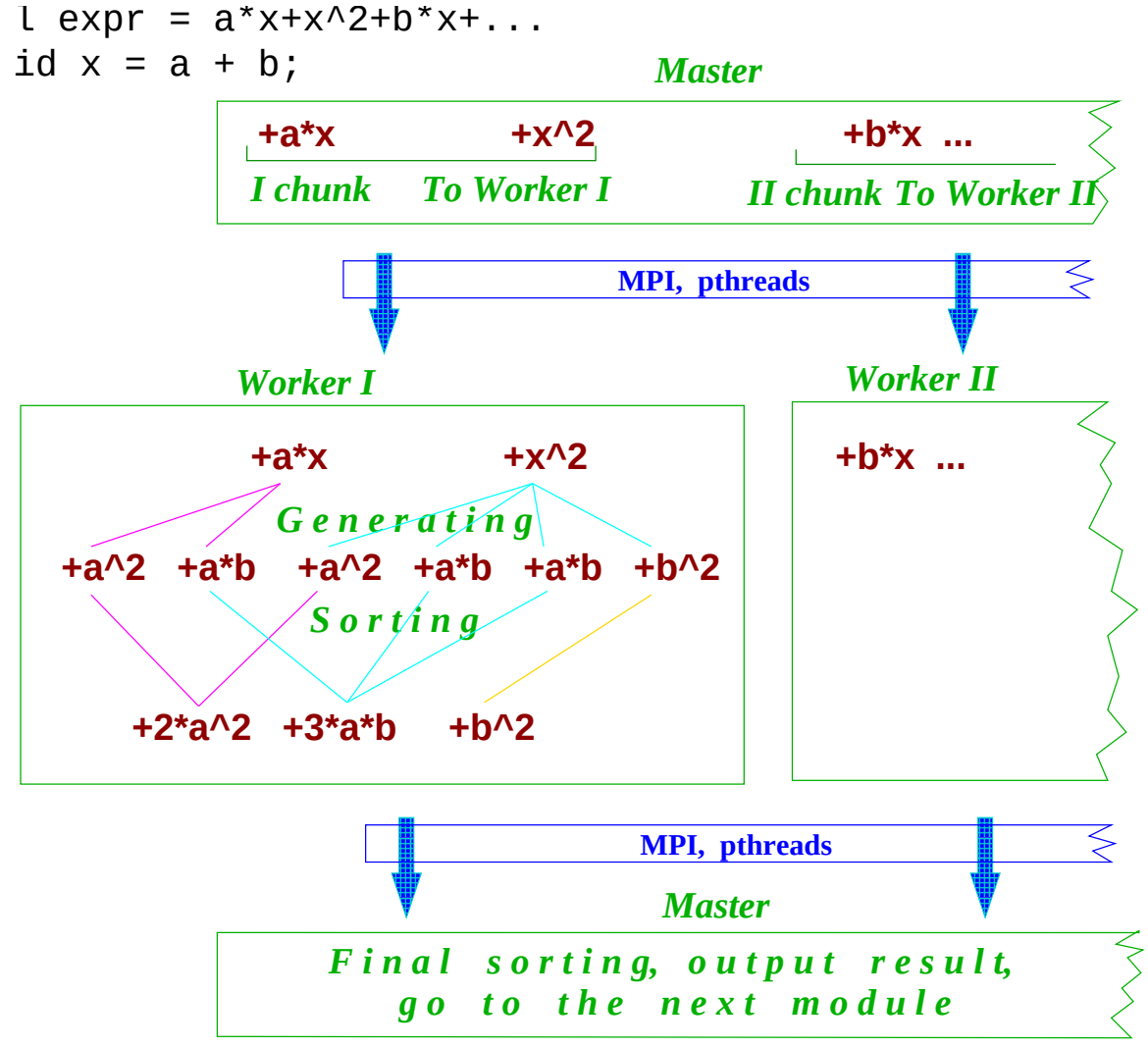
The master splits the input expression in chunks. Each chunk is sent to one of the worker. Workers perform generating/sorting and send the result back.



The concept of FORM parallelization

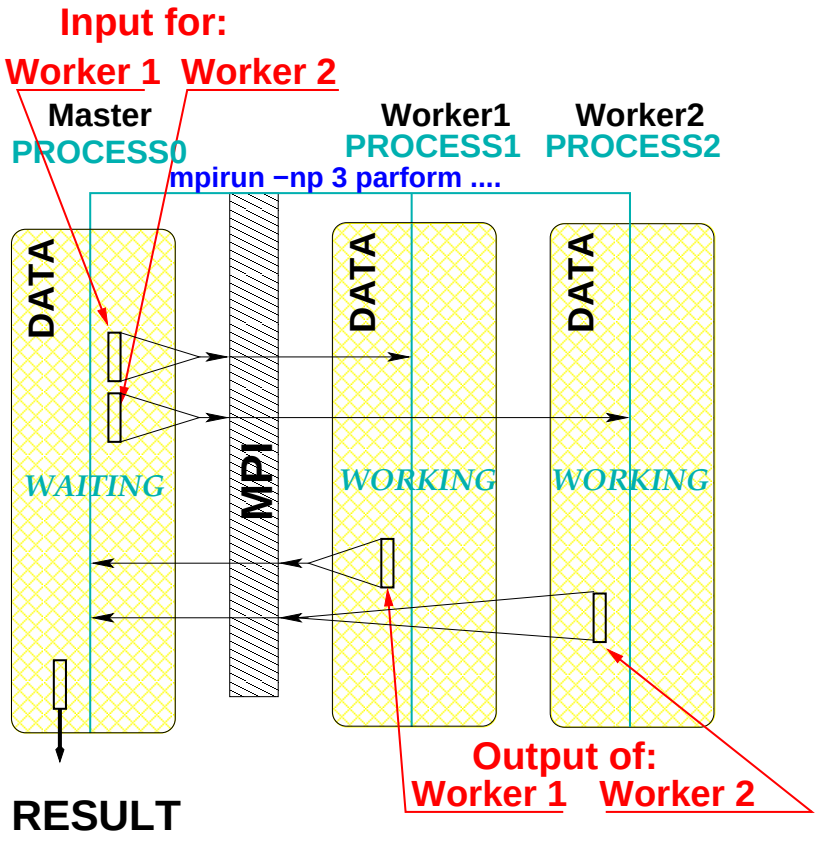
Transparently for the user! The same FORM program. No special efforts for parallel programming.

All FORM programs may be executed in parallel without any changings.

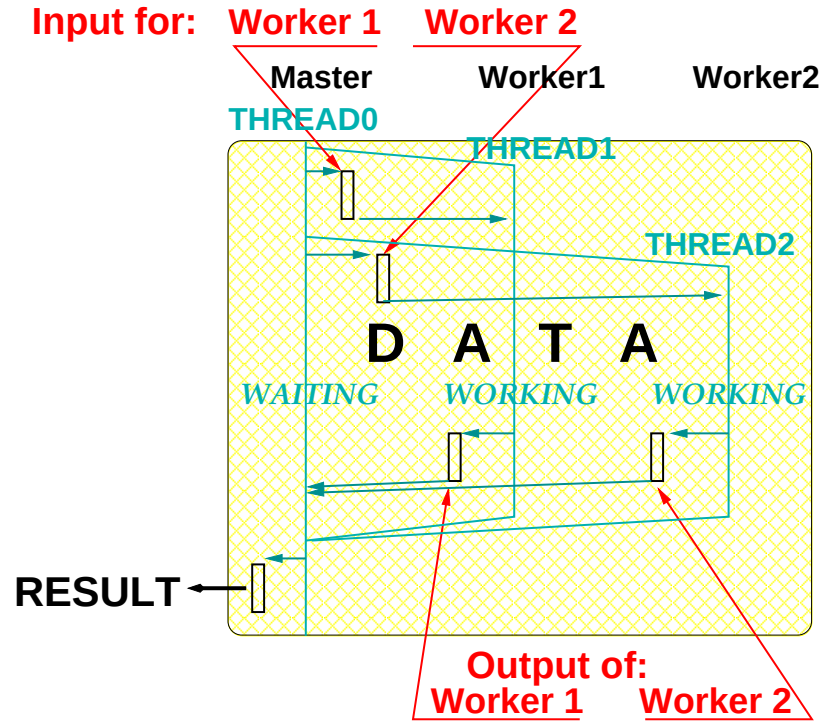


Models in use:

SMP and Clusters: MPI (Message Passing Interface)



SMP computer: Multithreaded processes



ParFORM: uses MPI
Karlsruhe, 1998

TFORM: uses POSIX Threads
NIKHEF, 2005

Advantages and disadvantages:

ParFORM: independent processes.
Individual computational nodes: clusters, Massive Parallel Processing (MPP)

Advantages and disadvantages:

ParFORM: independent processes.
Individual computational nodes: clusters, Massive Parallel Processing (MPP)

TFORM: common address space. Only SMP computers.

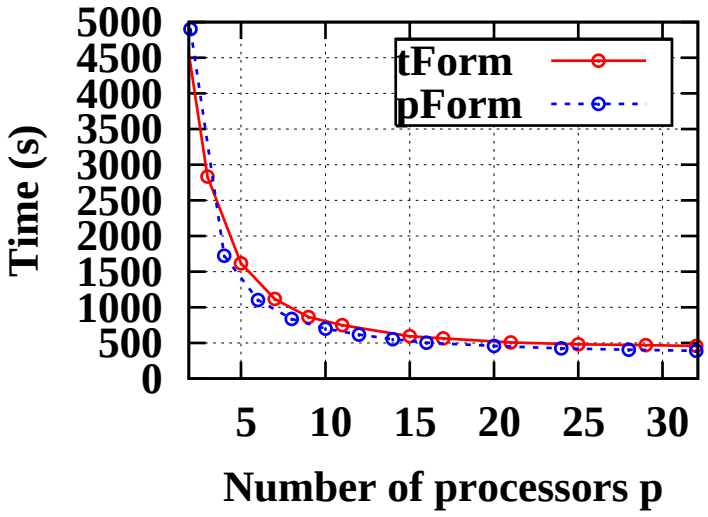
- ☞ No installation! Just load executable file and run it!
- ☞ Shared address space allows to implement some features which are hardly any possible for ParFORM.

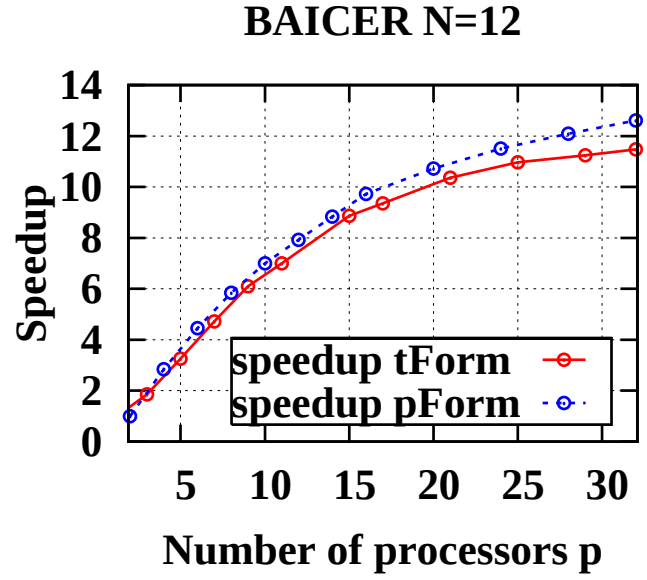
Advantages and disadvantages:

ParFORM: independent processes.
Individual computational nodes: clusters, Massive Parallel Processing (MPP)

TFORM: common address space. Only SMP computers.

Scalabilities are almost coincide.The right graph is a speedup curve

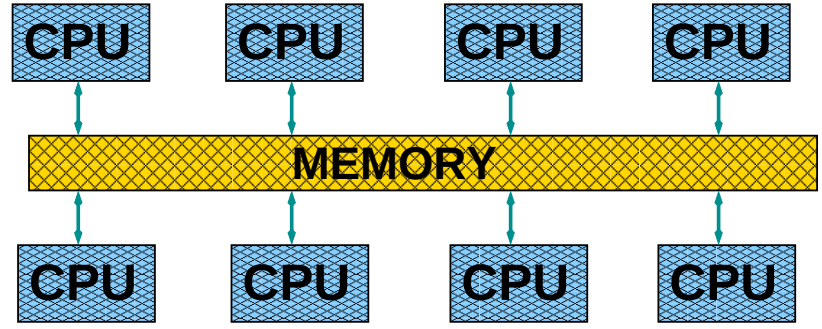




$$S(p) = \frac{T(1)}{T(p)}$$

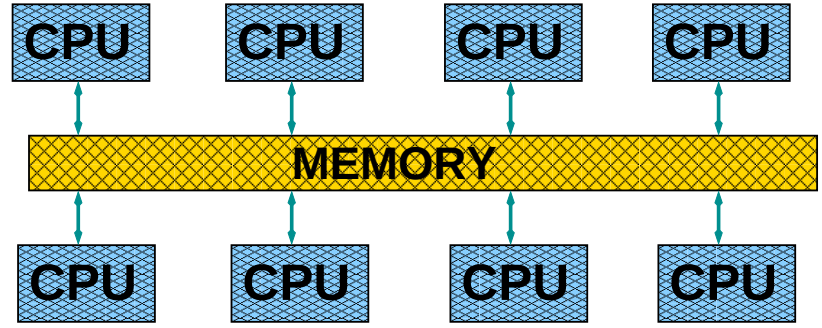
Possible hardware

SMP (Symmetric Multi Processing), several identical processors are connected to a single shared main memory.

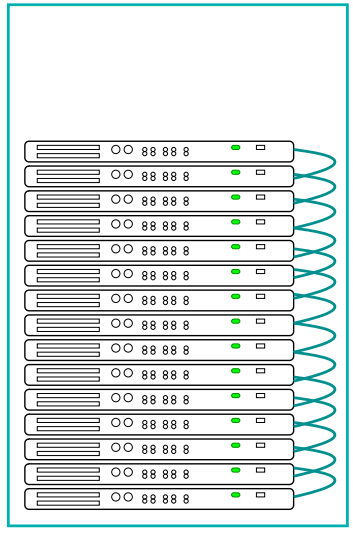


Possible hardware

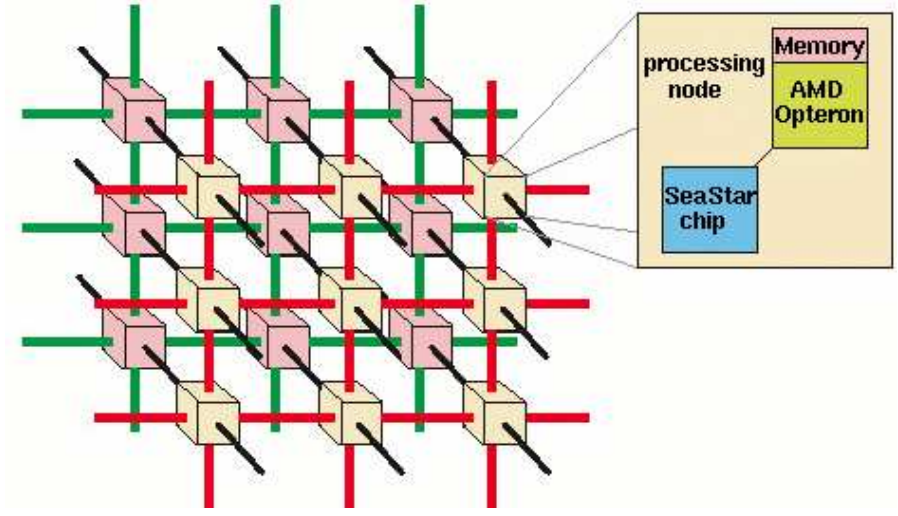
SMP (Symmetric Multi Processing), several identical processors are connected to a single shared main memory.



Clusters: several computers connected by some fast network.

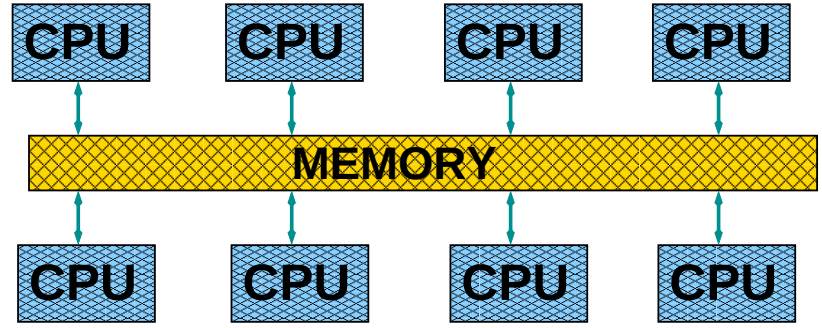


MPP (Massive Parallel Processing), e.g. CrayXT3:

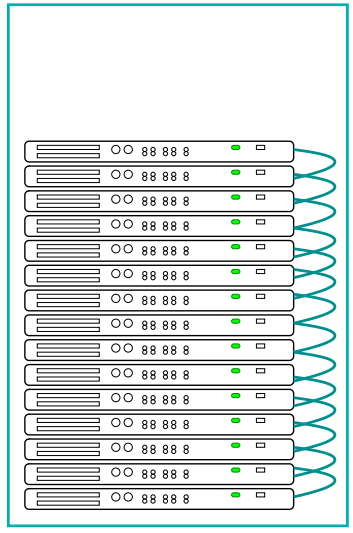


Possible hardware

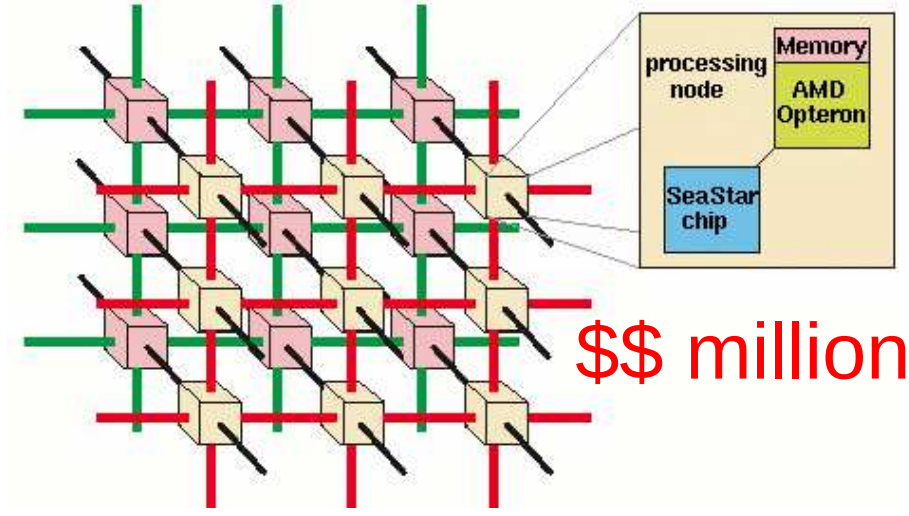
SMP (Symmetric Multi Processing), several identical processors are connected to a single shared main memory.



Clusters: several computers connected by some fast net-work.

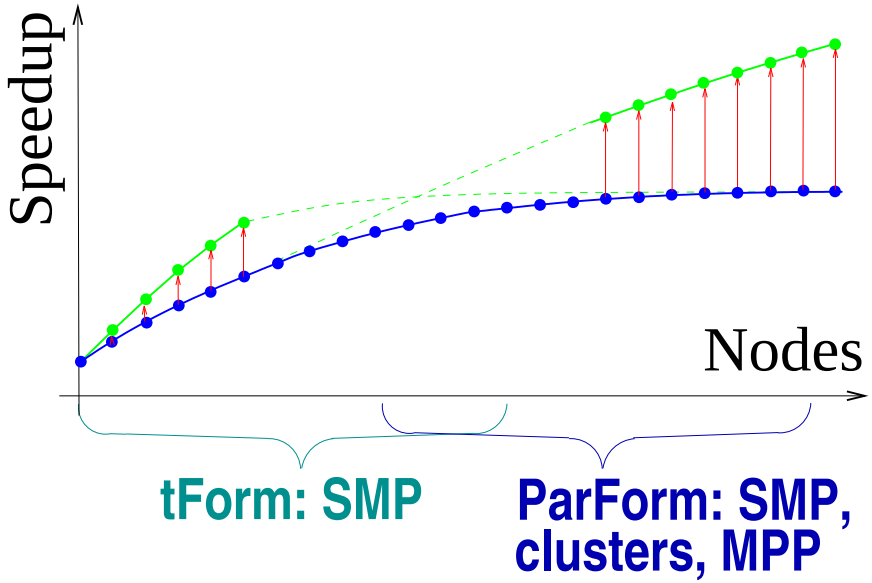


MPP (Massive Parallel Processing), e.g. CrayXT3:



$$ millions!

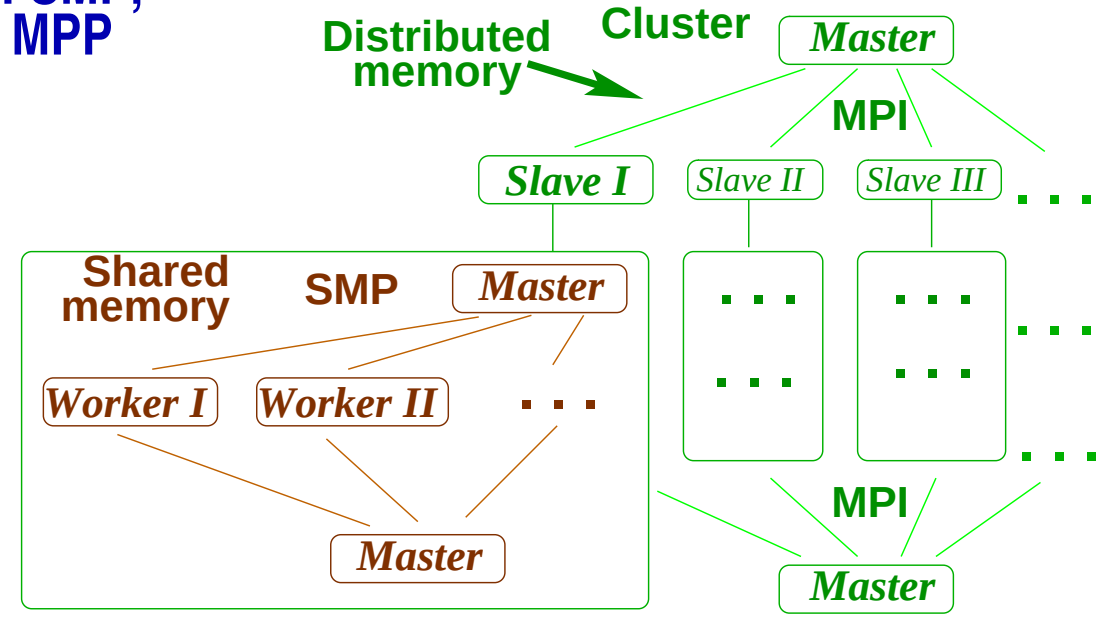
Parallelization: outlooks



Problems:

- Final sorting only by the Master.
- I/O traffic jams.

Combining Par-FORM and TFORM:



What FORM is missing?

Non-local operations:

- Non-local substitutions, e.g. `id a + b = c;`
- Non-locall summing up similar terms, especially denominators, e.g. `l e = a/(b+c) + d/(b+c);` leads to `e = 1/(c + b)*d + 1/(c + b)*a;`
- Factorization
- Greatest Common Divisor (GSD) contraction

Miscellaneous:

- Automatic setup parameters
- Local variables
- Open sources

etc...

Workarounds

Bracketing:

```
s a,b,c,d;
f acc;
l e = a/(b+c) + d/(b+c);
id 1/(b+c) = acc(1/(b+c));
b acc;
print;
.end
```

will result in

```
e =  + acc(1/(c + b)) * ( d + a );
```

Workarounds

PolyFun:

```
PolyFun acc;
Local F = 3*x^2*acc(1+y+y^2)+2*x^2*acc(1-y+y^2);
```

will result in

```
F = x^2*acc(5+y+5*y^2);
```

PolyRatFun:

```
PolyRatFun acc;
Local F = 3*x^2*acc(1+y+y^2,1-y)+
          2*x^2*acc(1-y+y^2,1+y);
```

will result in

```
F = x^2*acc(-y^3-10*y^2-2*y-5,y^2-1);
```

"Poly_" functions

"Functions":
polyadd_, polysub_, polydiv_, polymul_, polygcd_

```
 s x1,x2,x3,x4;
    cf f;
    l e = f(x2);
    $a=x1^2+3*x1;
    $b=x1;
    id f(x2?)= polygcd_($a,$b);
    print;
    .end
 e =
       x1;
```