Parallel versions of the symbolic manipulation system FORM

Mikhail Tentyukov

TTP Karlsruhe

FORM, ParFORM and TFORM

FORM is a program by J. Vermaseren for symbolic manipulation of algebraic expressions specialized to handle very large expressions of millions of terms, http://www.nikhef.nl/˜form.

ParFORM is a parallel version of FORM developed in Karlsruhe. http://www-ttp.particle.uni-karlsruhe.de/˜parform

TFORM is a parallel version of FORM optimized for multicore computers developed by J. Vermaseren.

Recent development NIKHEF Amsterdam: J. Kuipers, I. Pushkina, J. Vermaseren, J. Vollinga

TTP Karlsruhe:

J. Kühn, M. Steinhauser, M. Tentyukov

FORM setup

Module by module. Each module:

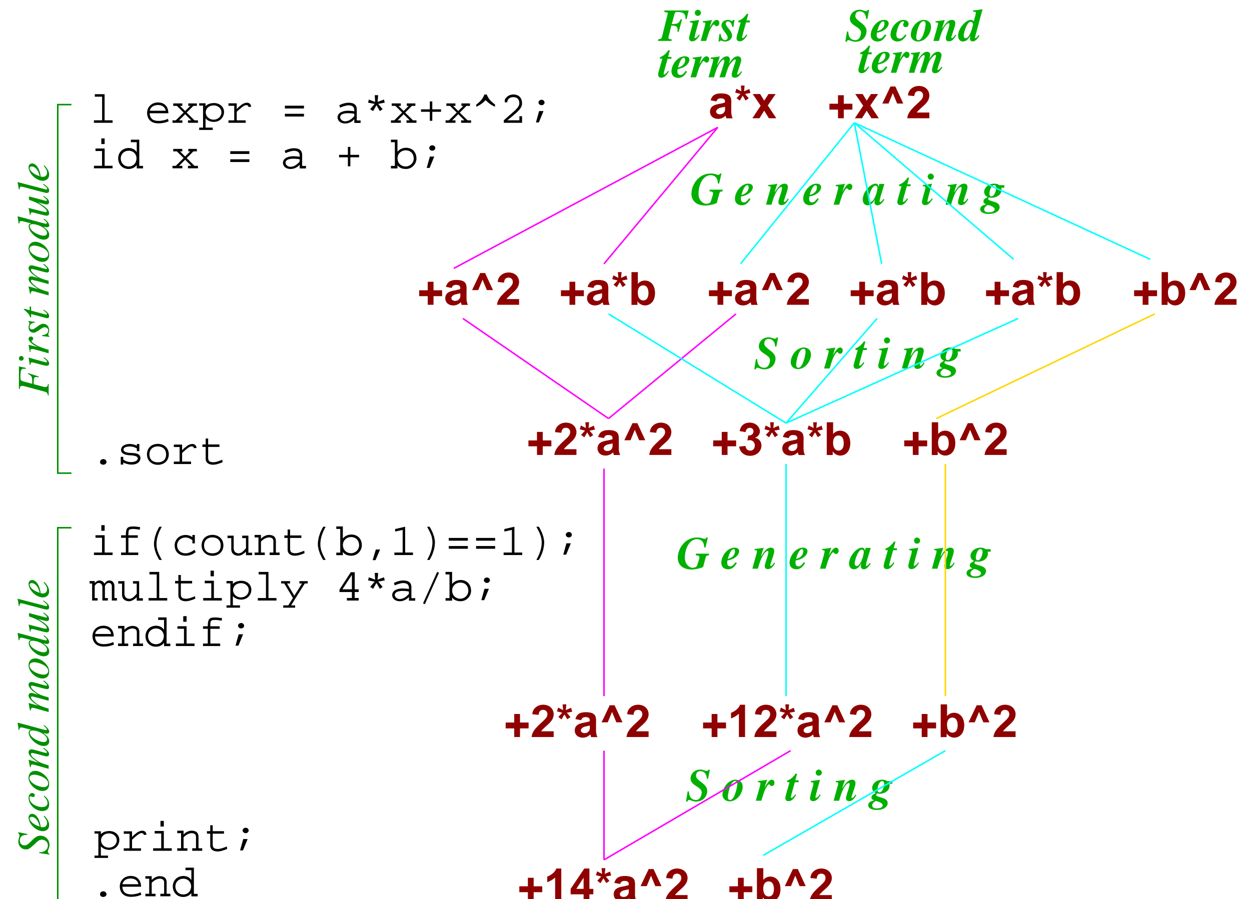
1. Compilation. 2. Generating. 3. Sorting:

Very long expressions!

Non-interactive interpreter.

User provides a program, Interpreter runs the program.

Modules are terminated by "dot" instructions.



FORM setup

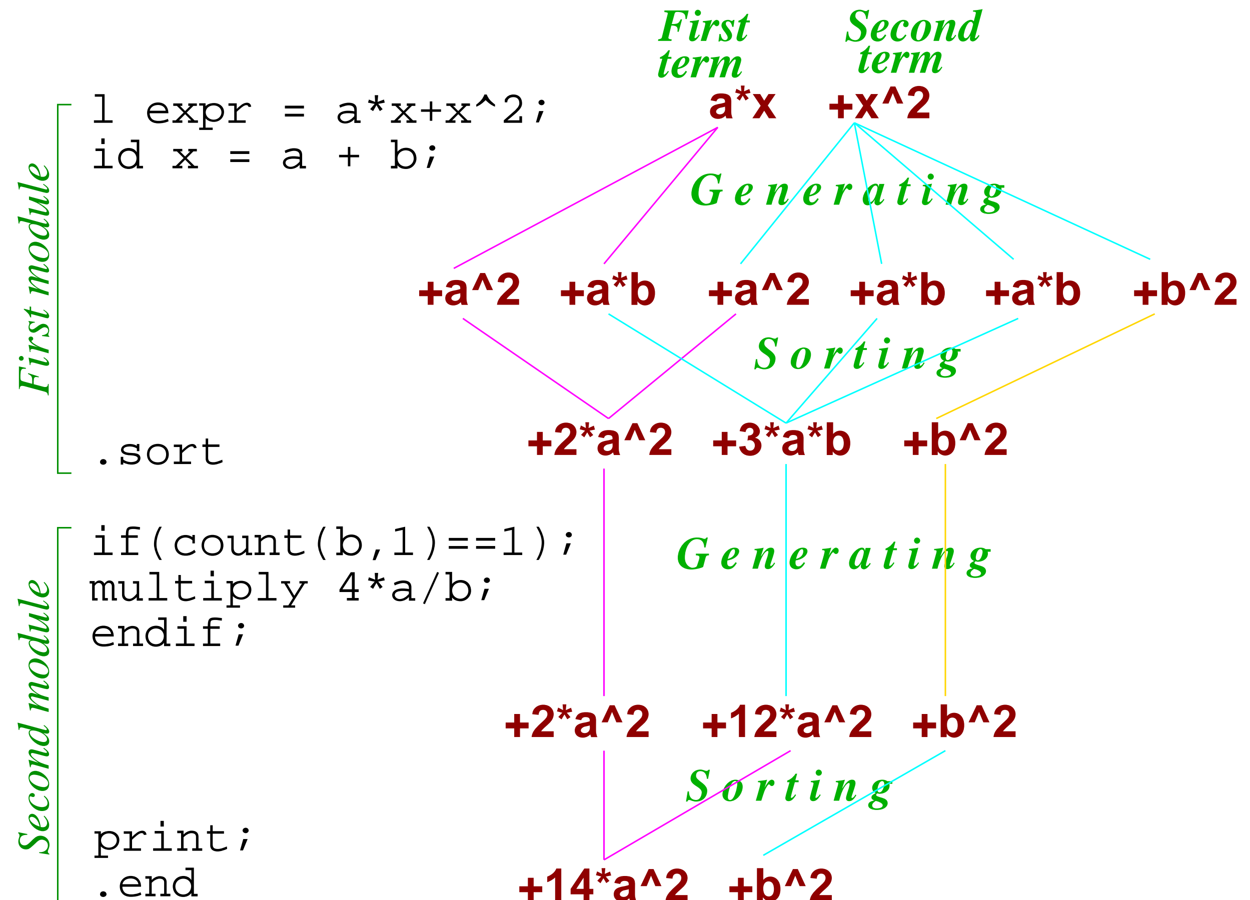
Module by module. Each module:

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

Each module:

Definition of new expressions, algebraic instructions, output instructions.

Expressions remain active through many modules.



FORM setup

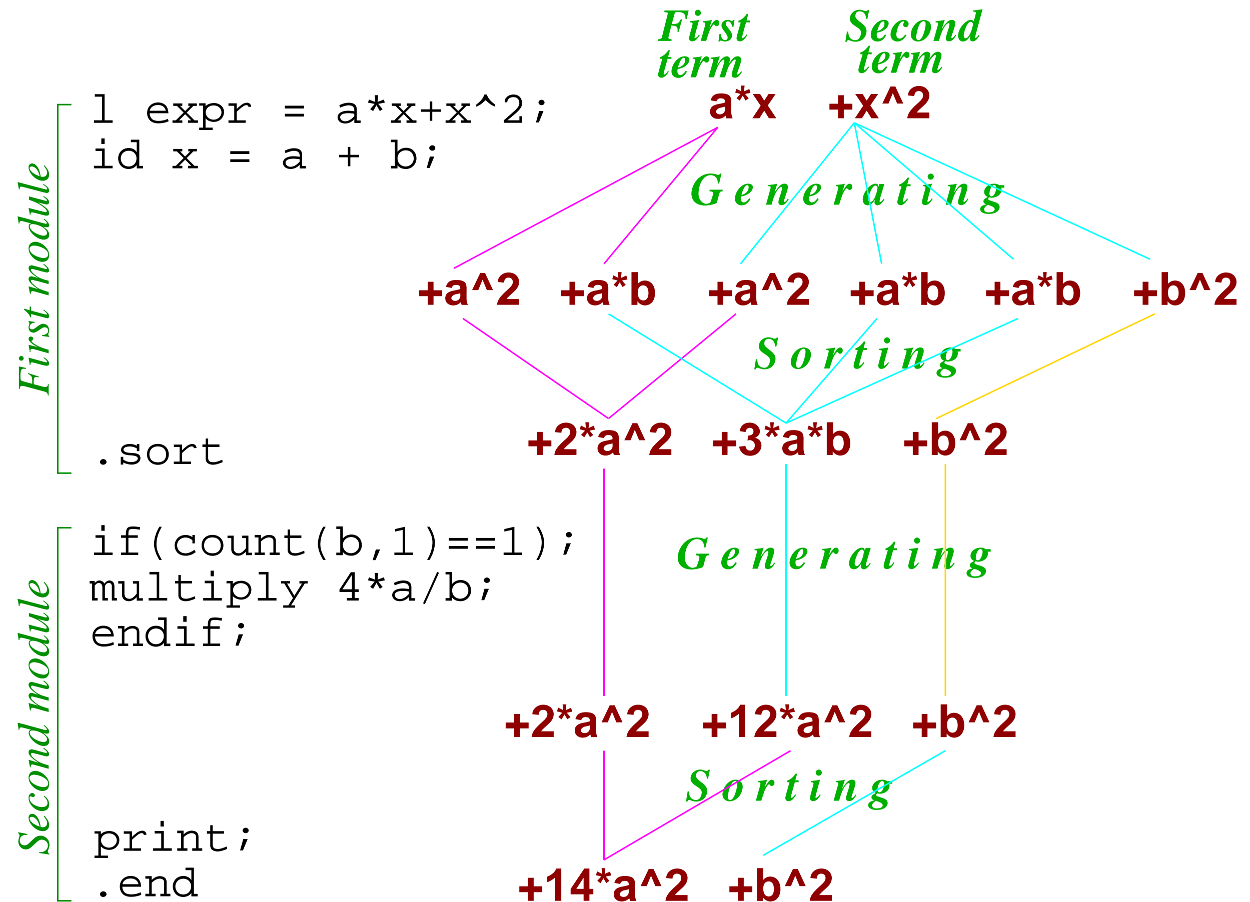
Module by module. Each module:

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 setup

Module by module. Each module:

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

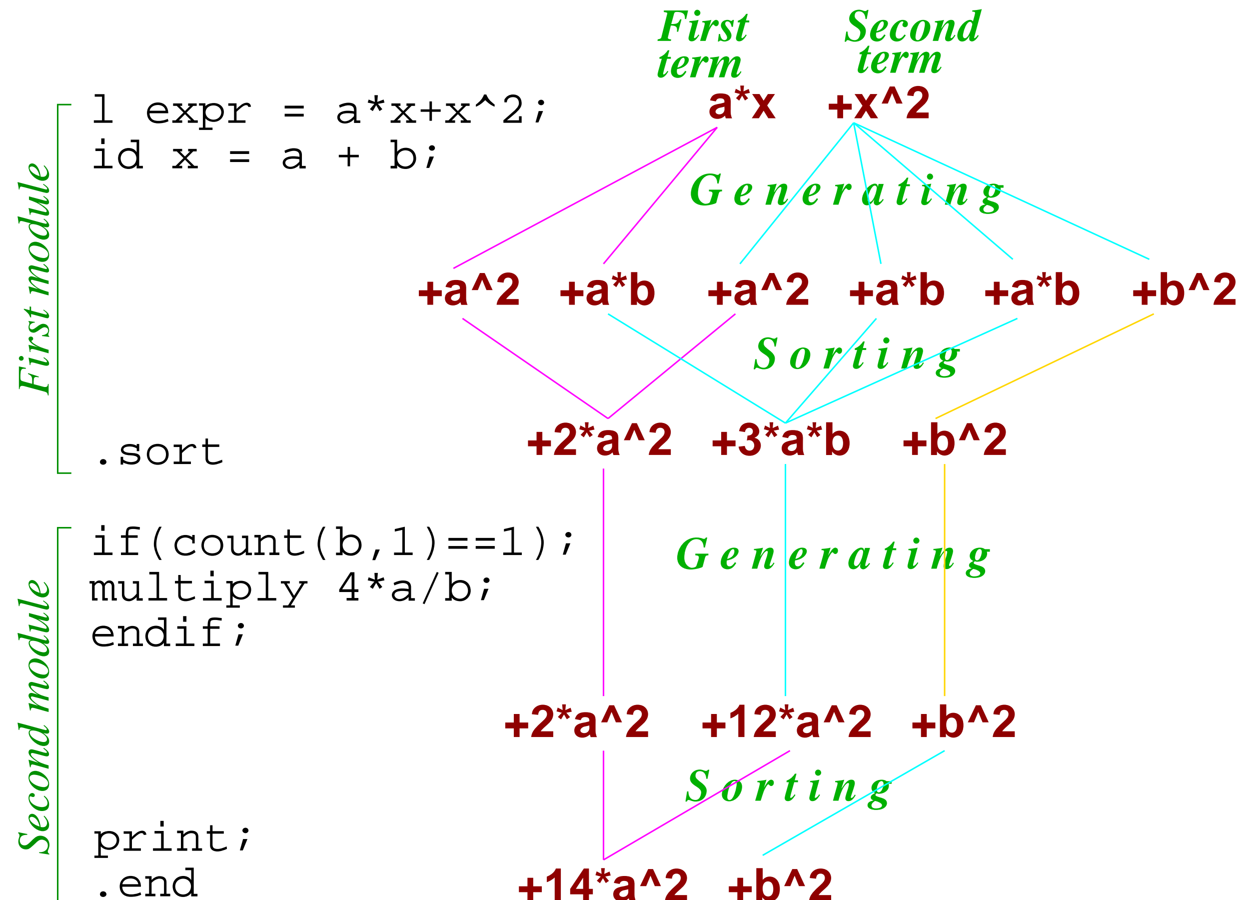
specific feature:

only *local* operations on single term:

id x = a + b;

Non-local operations are not allowed:

~~id a + b = x~~;

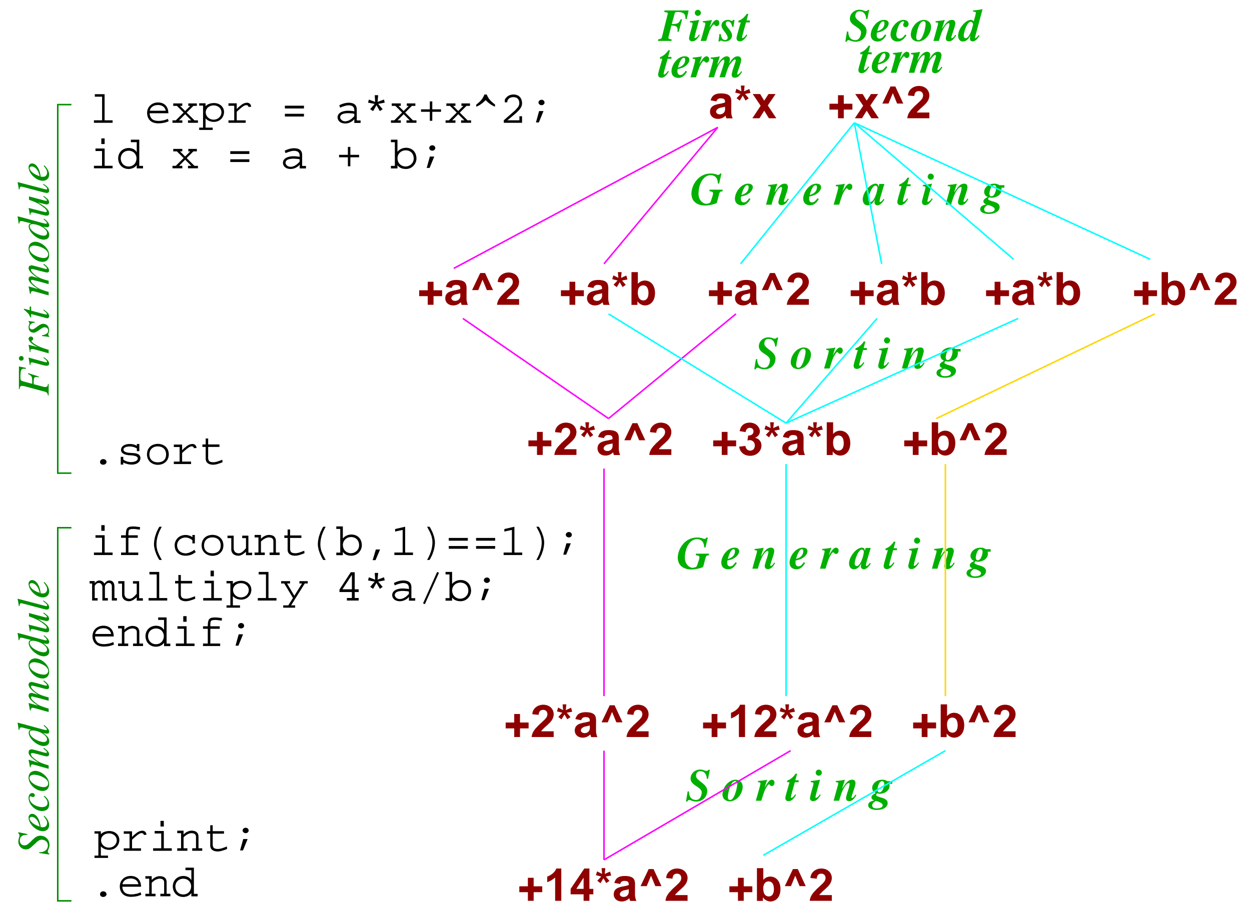


FORM setup

Module by module. Each module:

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

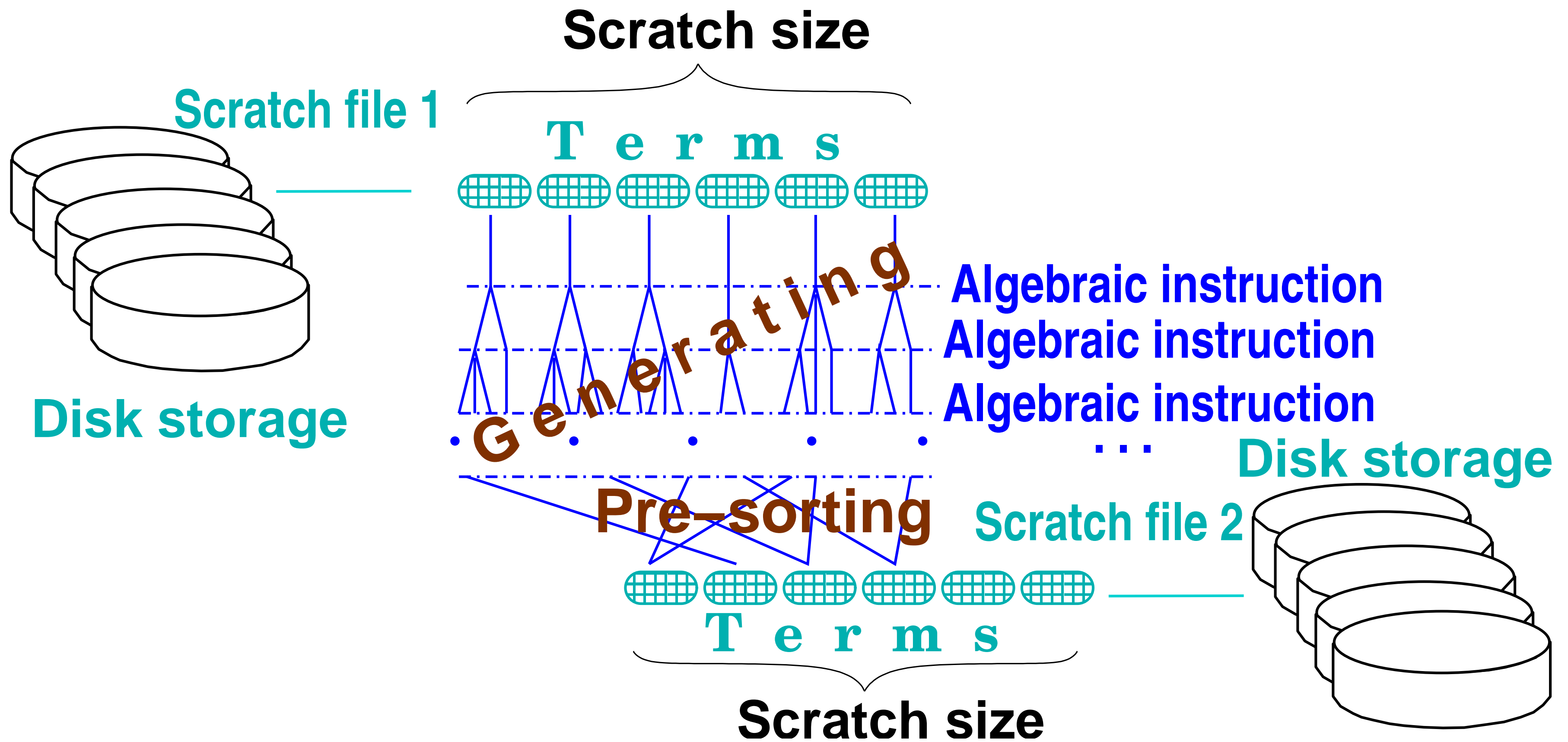
Locality Principle: All *explicit* algebraic operations are local. Non-local operations are allowed only *implicitly* in the sorting procedure at the end of the modules and in some other special cases.



Main FORM feature

Locality Principle → Expressions as “streams” of terms

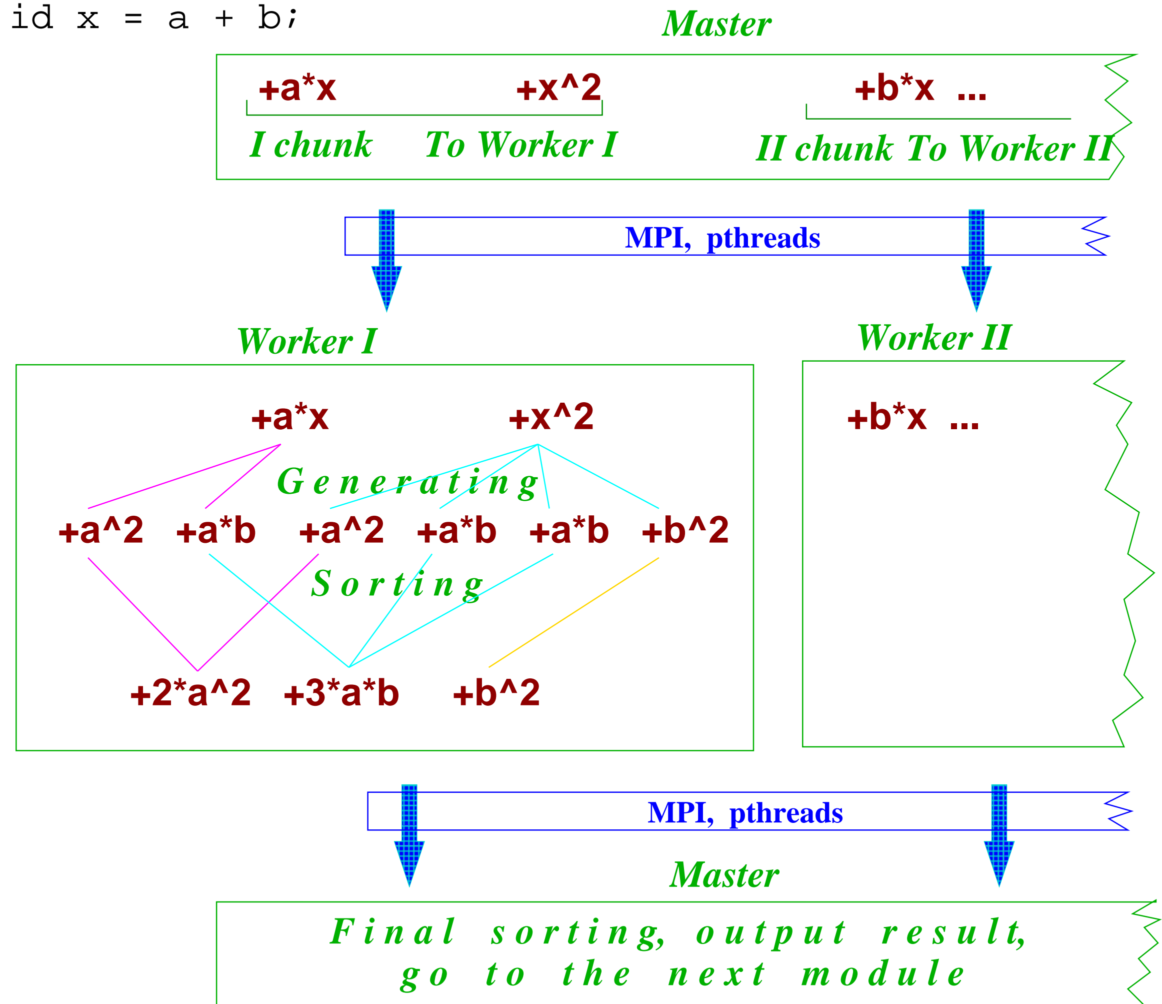
Expressions bigger than the memory (RAM) available!



The concept of FORM parallelization

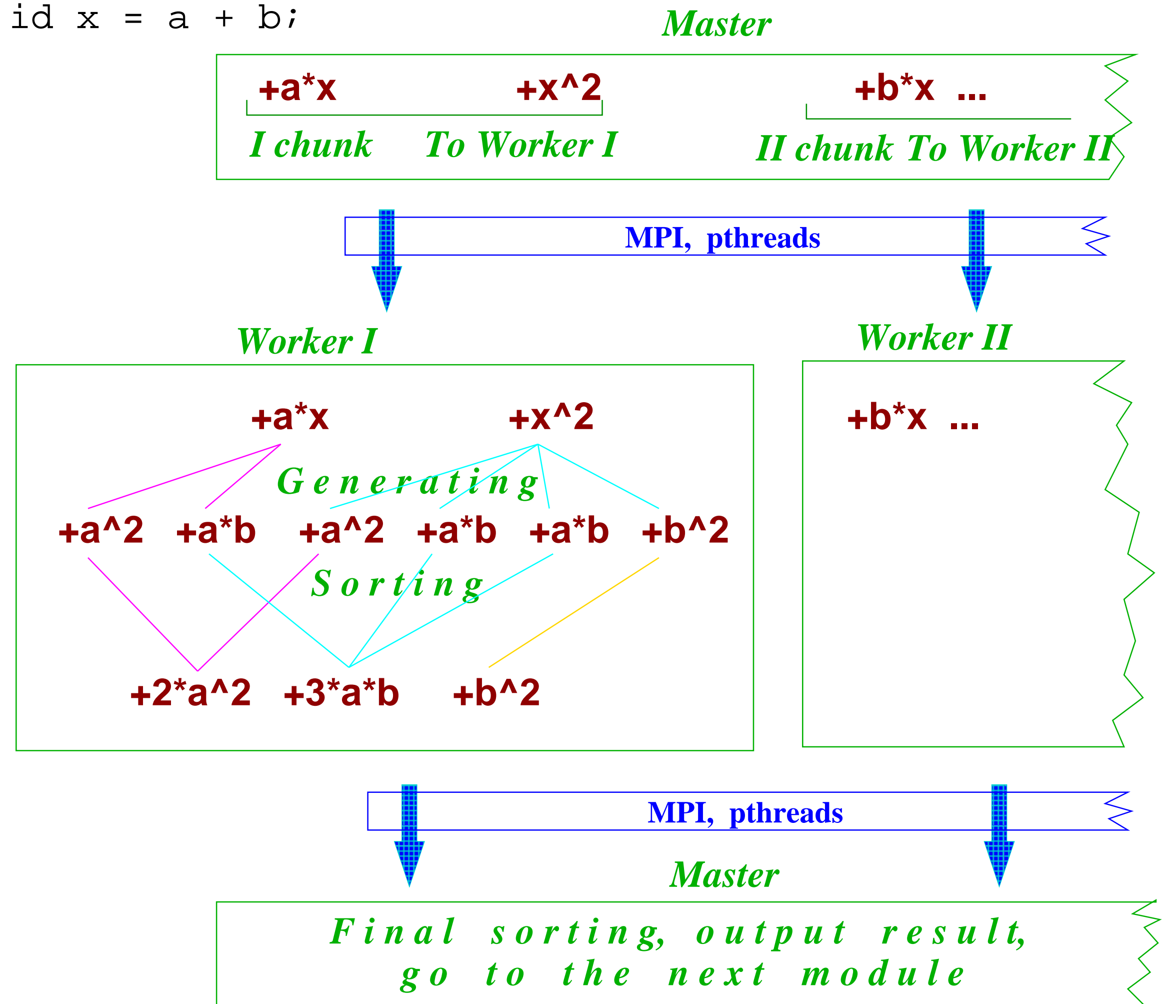
The *Locality Principle*, again!

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.



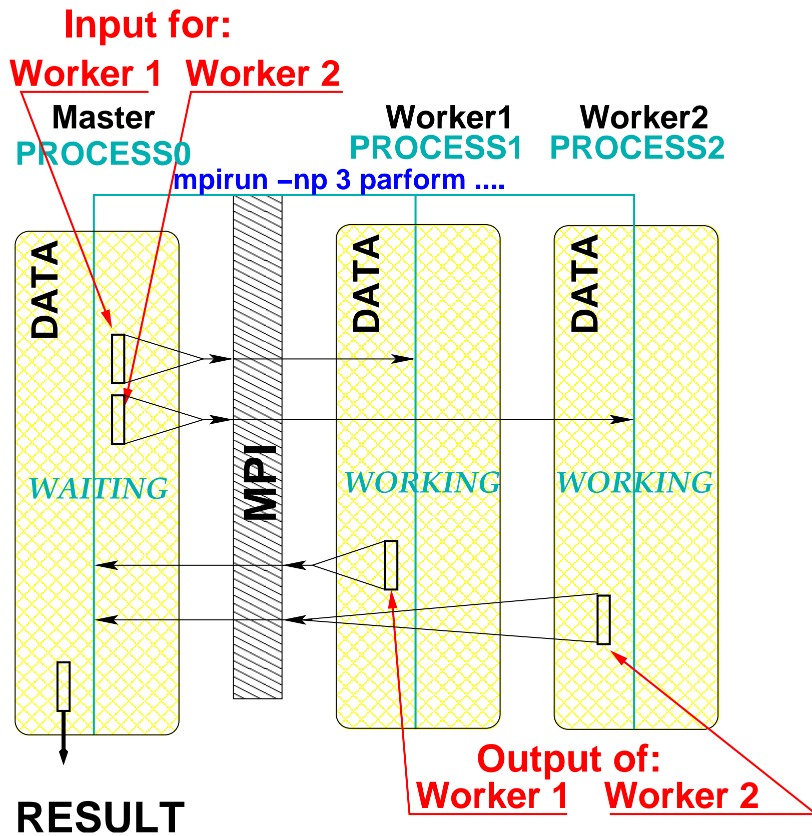
The concept of FORM parallelization

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.



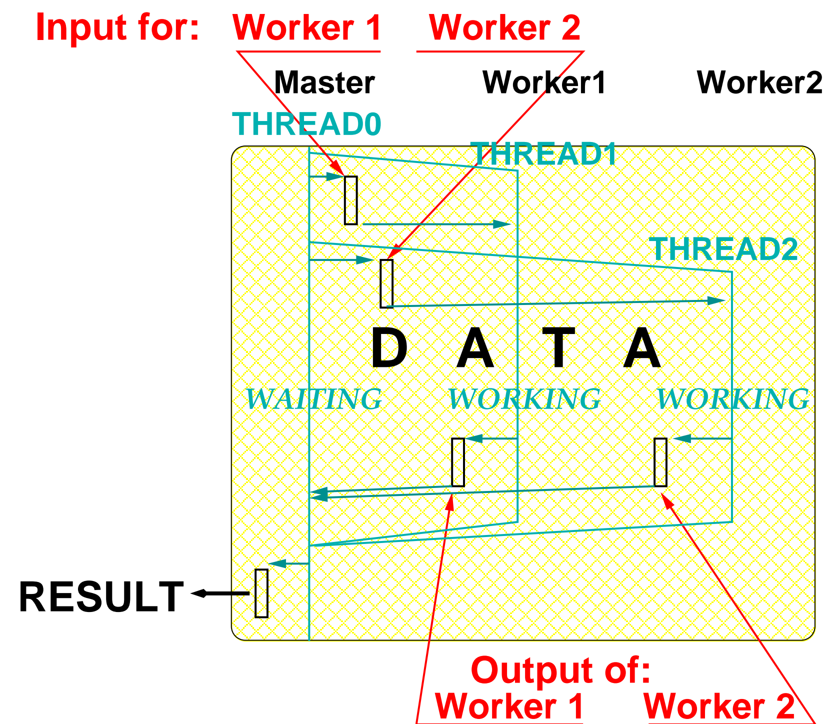
Models in use:

SMP and Clusters: MPI
(Message Passing Interface)



ParFORM: uses MPI

SMP computer:
Multithreaded processes



TFORM: uses POSIX Threads

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.

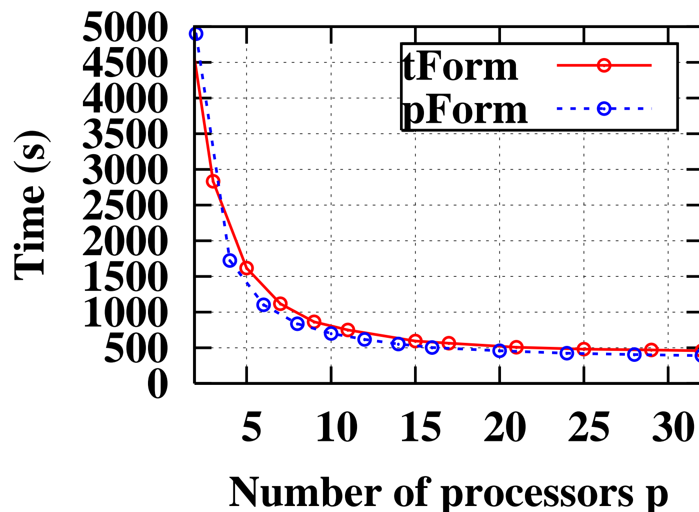
- ☞ No installation! Just load executable file and run it!
- ☞ Shared address space allows to implement some features which are hardly any possible for ParFORM.

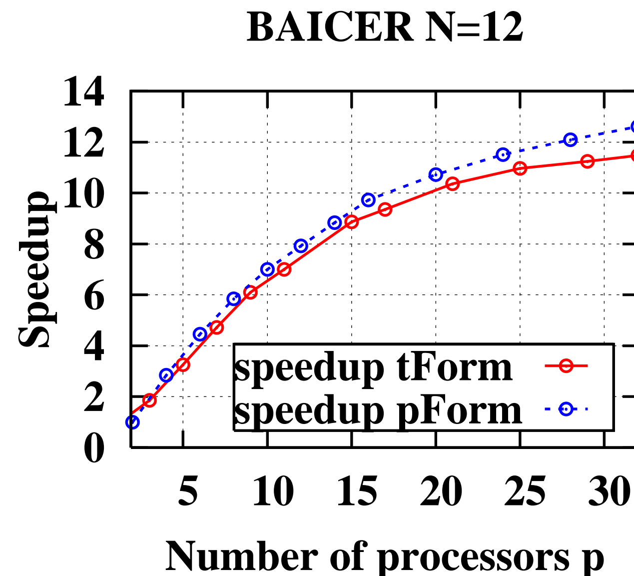
Advantages and disadvantages:

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

TFORM: common address space. Only SMP computers.

Scalabilities are almost coincide. Left graph – the speedup curve





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

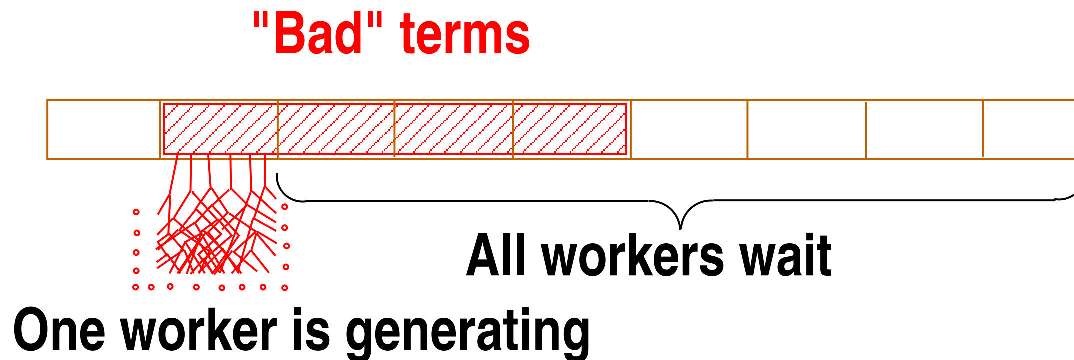
Load balancing

Efficient technique of load balancing: the Master immediately sends next chunk to ready worker.

The smaller chunk the better load balancing but the worse performance. With big chunk, it is easy to run into the worst case.

Load balancing

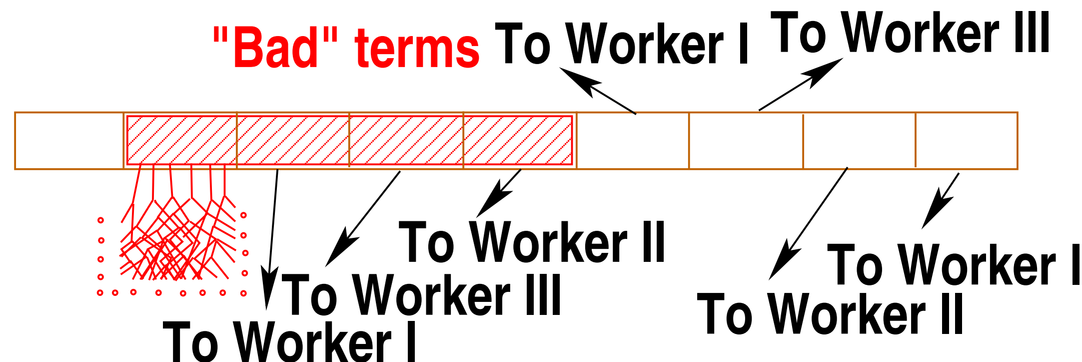
Efficient technique of load balancing: the Master immediately sends next chunk to ready worker.
Problem with "bad" terms:



The "bad" terms produces a lot of new terms.

Load balancing

Efficient technique of load balancing: the Master immediately sends next chunk to ready worker.
Problem with “bad” terms:



The idea: steal the tail and re-distribute it among free workers. By default is on; can be switched on or off with the statements:

on ThreadLoadBalancing;
off ThreadLoadBalancing;

} TFORM only!

Inparallel mode

The terms of expressions are distributed over the workers and the expressions are executed one by one.
When there are many small expressions, it is useful to execute each expression on its own processor.

Specification statements:

```
inparallel <list of expressions>;
notinparallel <list of expressions>;
```

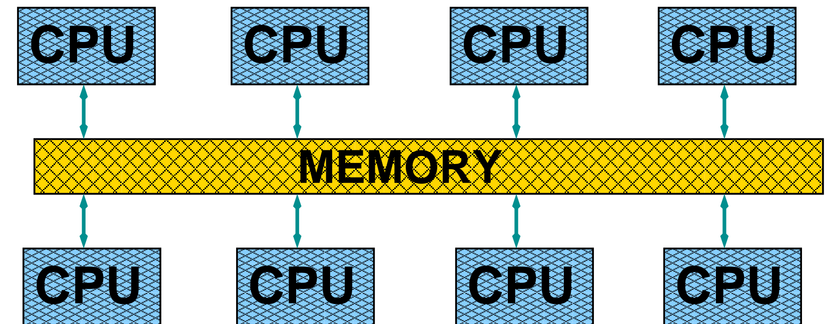
One should be careful using this statement for big expressions! Typical usage:

```
InParallel;
NotInParallel F1,F25;
```

would first mark all expressions to be executed in simultaneous mode and then make an exception for F1 and F25.

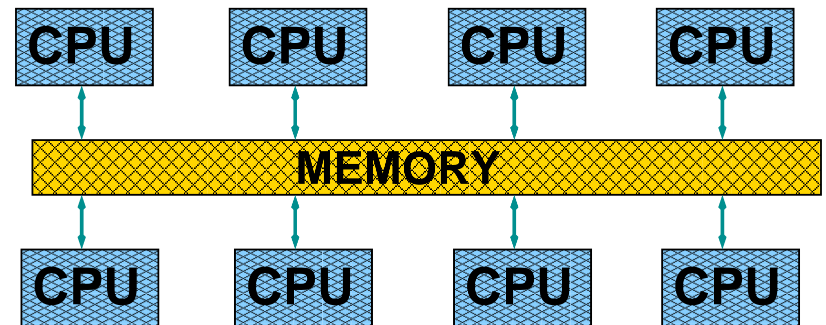
Possible hardware

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

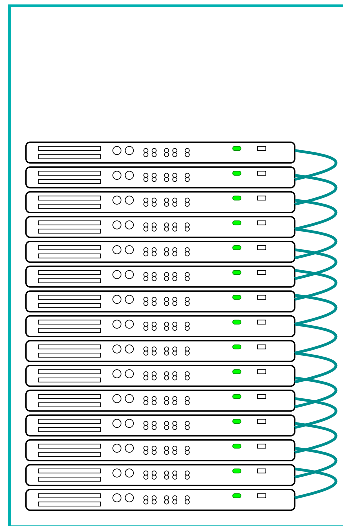


Possible hardware

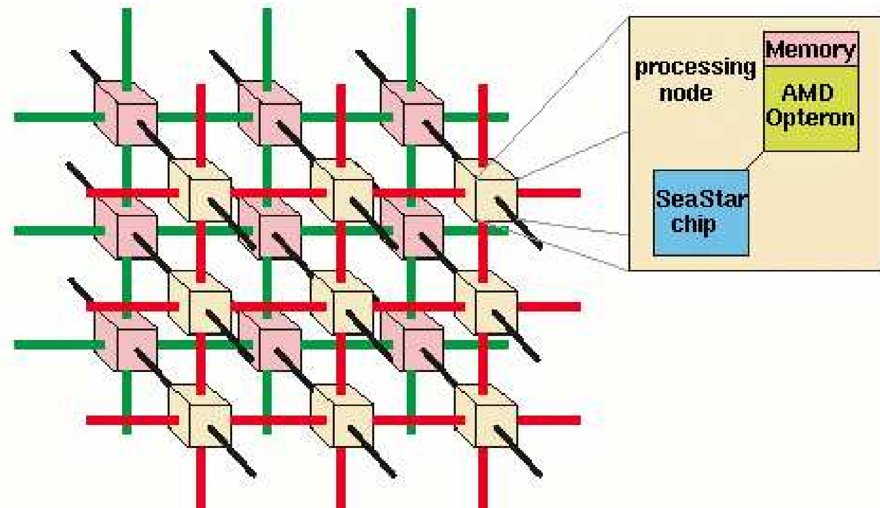
SMP (Symmetric Multi Processing), several identical processors are connected to a single shared main memory.



Clusters: several computers connected by some fast network.

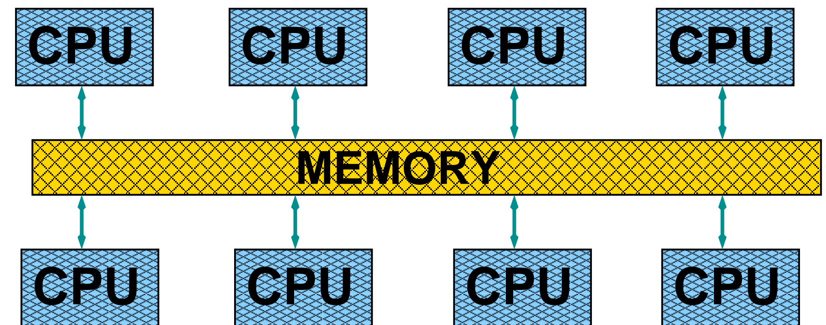


MPP (Massive Parallel Processing), e.g. CrayXT3:

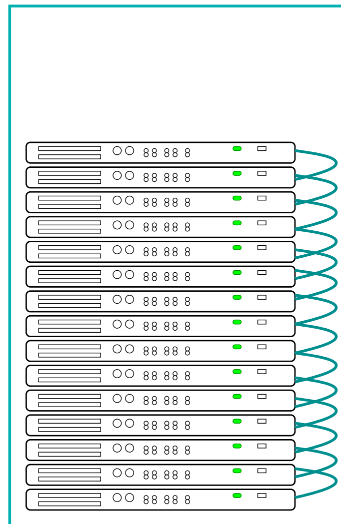


Possible hardware

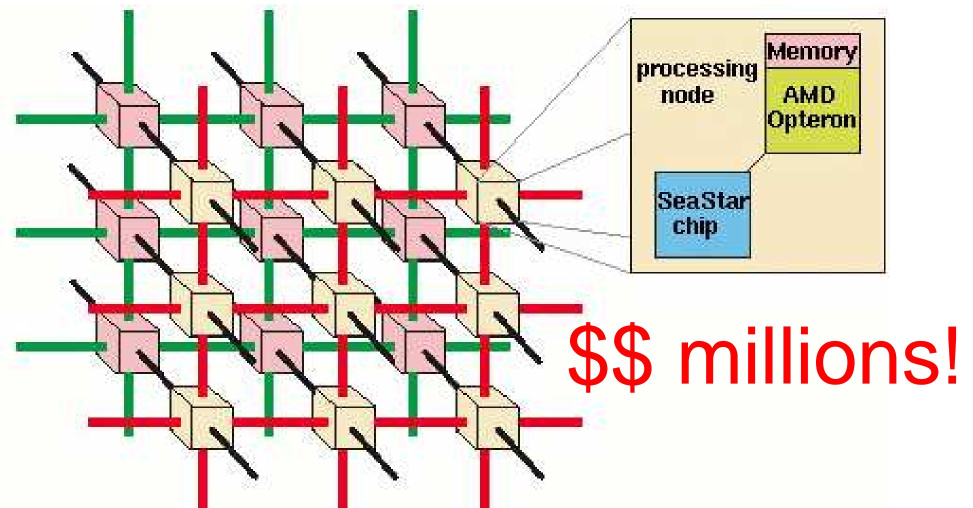
SMP (Symmetric Multi Processing), several identical processors are connected to a single shared main memory.



Clusters: several computers connected by some fast network.

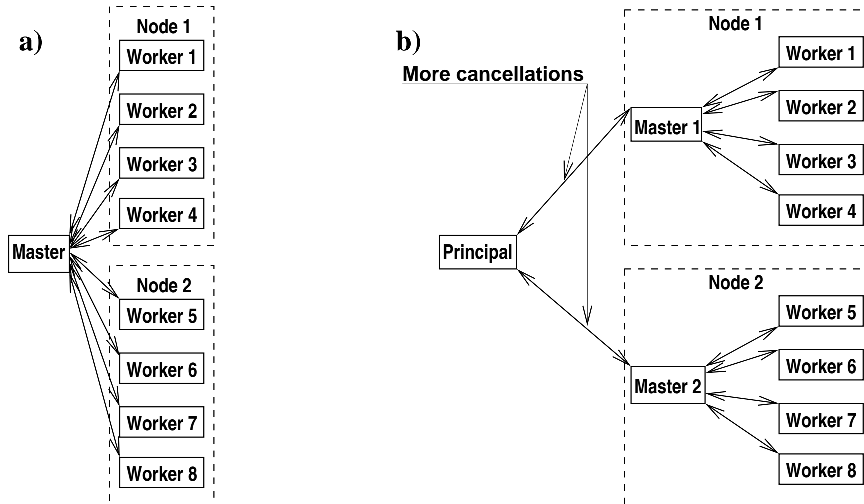


MPP (Massive Parallel Processing), e.g. CrayXT3:



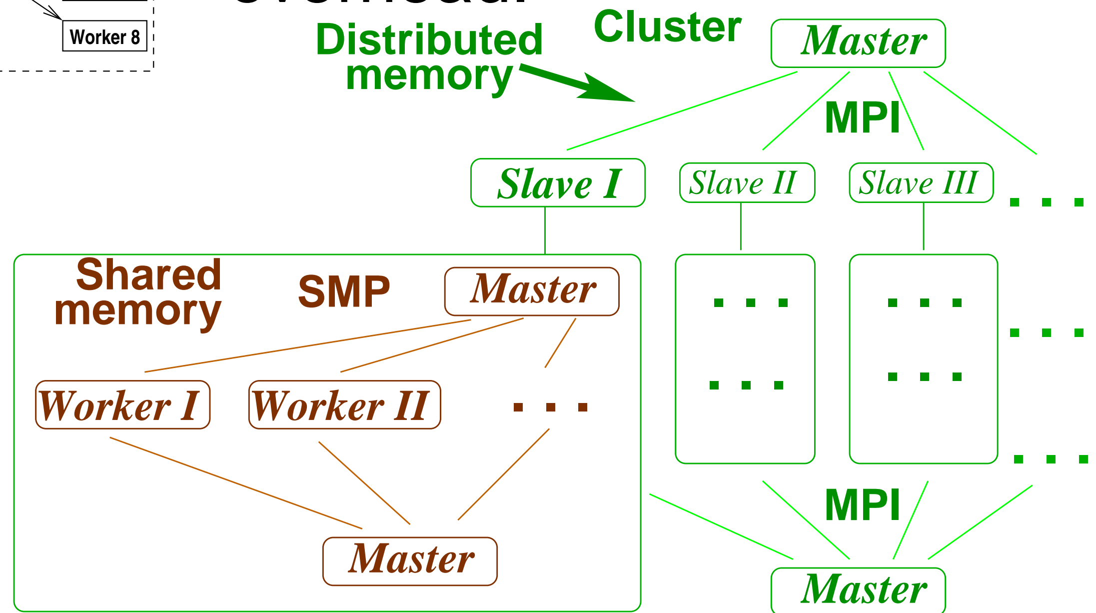
$$ millions!

Outlooks



Multicore clusters: the "star" topology leads to big network traffic.
Master on each node: MPI overhead.

Combining ParFORM and TFORM:



Conclusion

- ☞ Both ParFORM and TFORM are able to execute almost all FORM programs in parallel.
- ☞ ParFORM supports more hardware architectures. TFORM supports parallelization of more FORM features.
- ☞ ParFORM requires MPI, TFORM doesn't: much easy to deploy.
- ☞ TFORM is optimal for parallelization on small (≤ 8) number of CPUs. ParFORM is optimal for parallelization on large (≥ 6) number of CPUs.
- ☞ In future, ParFORM and TFORM will be combined to get advantages of each of the approaches.