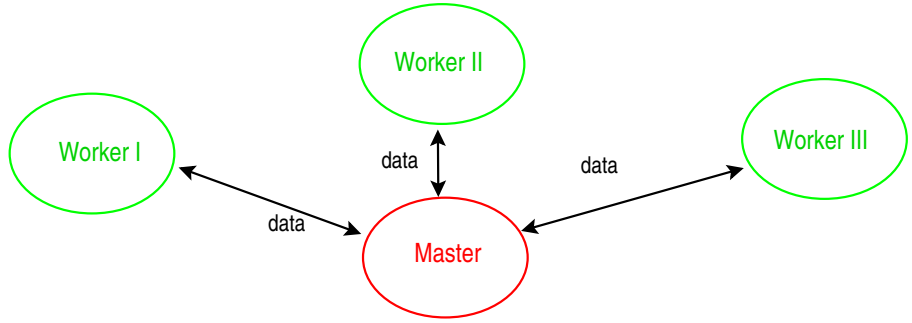
Current status of FORM parallelization

M. Tentyukov TTP Karlsruhe
J.A.M. Vermaseren NIKHEF

November 3-7, 2008

General concept

Master splits expressions into chunks. Each chunk is sent to one Worker. Workers generate terms, sort them and send sorted terms back to the master. Master performs final sorting.

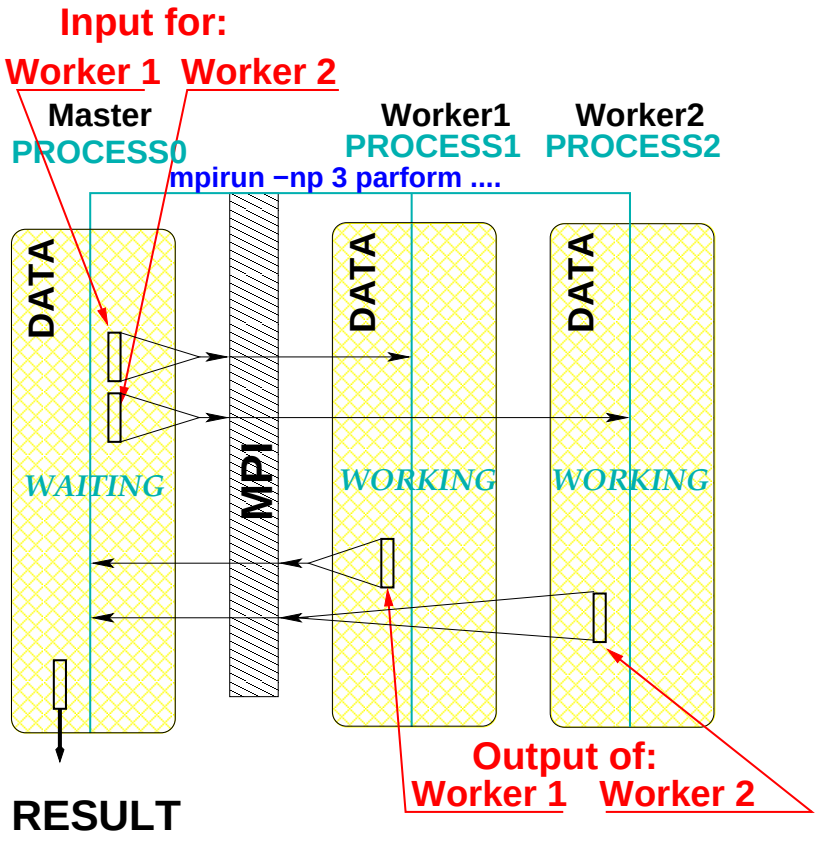


$$\texttt{Local expr} = \underbrace{\texttt{a\^{}2} + \texttt{a} * \texttt{x}}_{To\ Worker\ I} \quad \underbrace{+ \texttt{x\^{}3} - \texttt{b} * 2}_{To\ Worker\ II} + \underbrace{\ldots}_{To\ next\ workers}$$

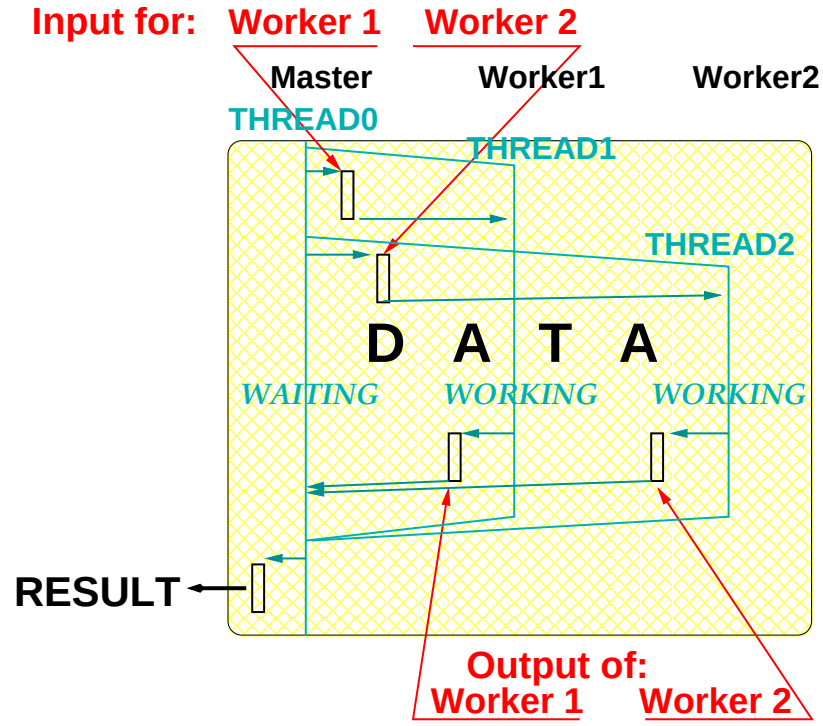
No special efforts for parallel programming!
The same FORM program runs in parallel!

Models in use:

SMP and Clusters: MPI (Message Passing Interface)



SMP computer: Multithreaded processes



ParFORM: uses MPI
Karlsruhe, 1998

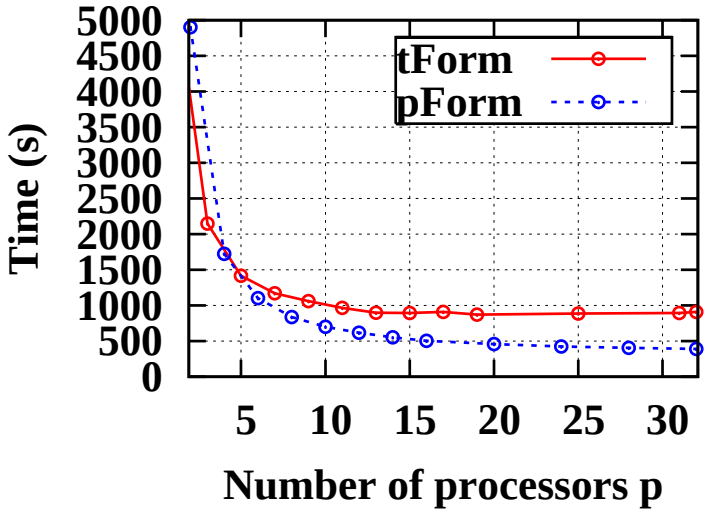
TFORM: uses POSIX Threads
NIKHEF, 2005

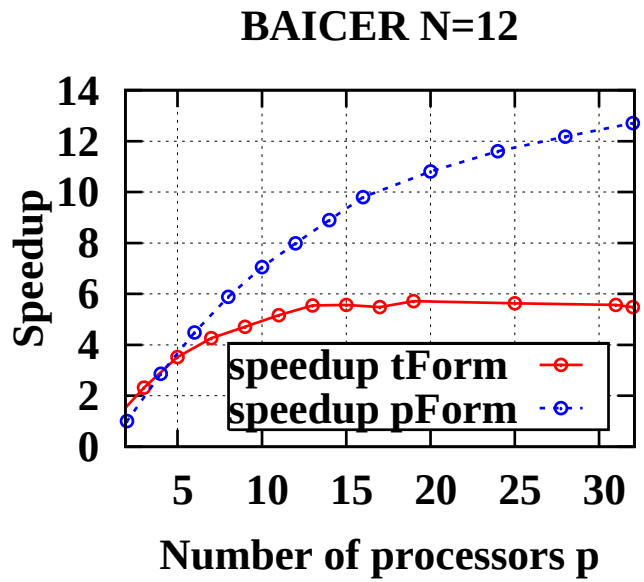
Advantages and disadvantages:

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. Less reliability and robustness, sometimes less scalability. Early version:

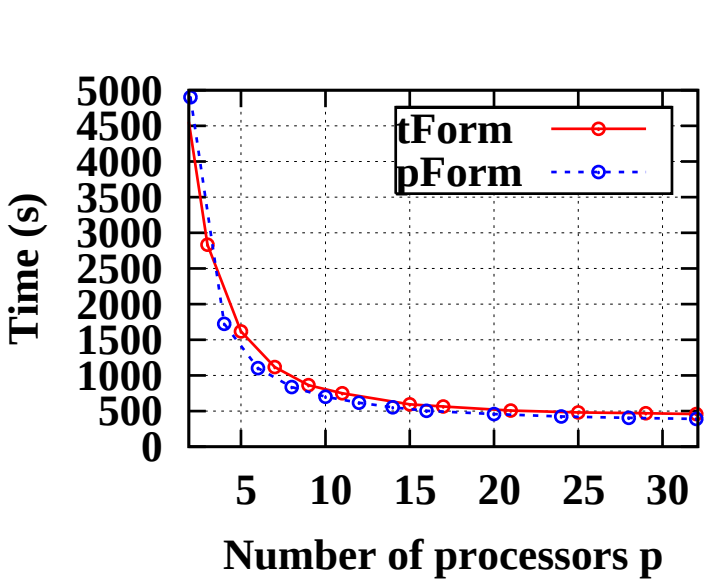


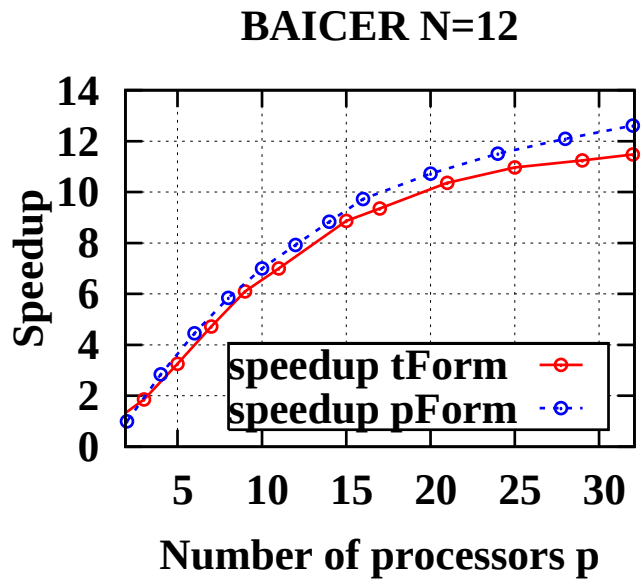


Speedup curve: $\frac{T_{seq}}{T_n}$.

Advantages and disadvantages:

ParFORM: independent processes.
Individual computational nodes: clusters, Massive Parallel Processing (MPP)
TFORM: common address space. Less reliability and robustness, sometimes less scalability. But now:





Scalabilities are almost coincide.

Advantages and disadvantages:

ParFORM: independent processes.
Individual computational nodes: clusters, Massive Parallel Processing (MPP)
TFORM: common address space. Less reliability and robustness, sometimes less scalability.

But:

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

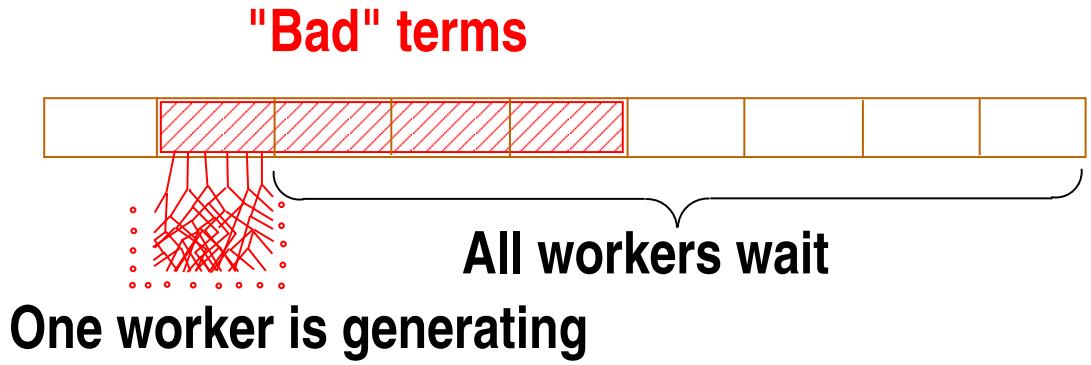
Load balancing

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

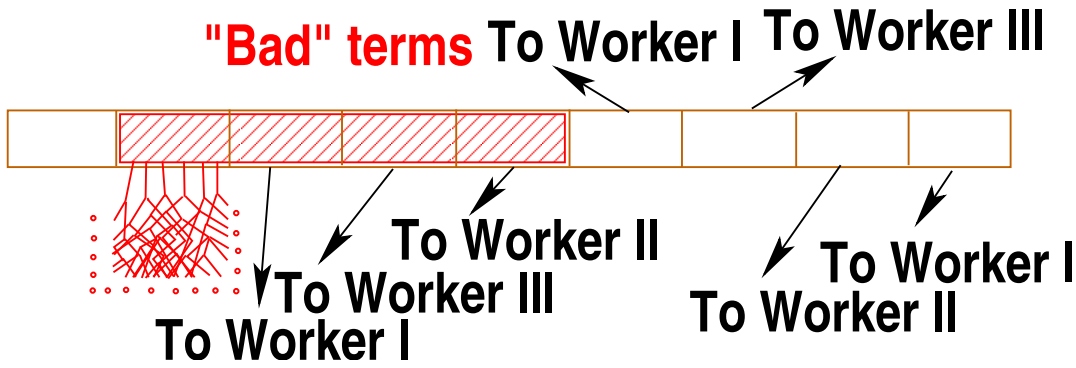
Efficient technique of load balancing: the Master immediately sends next chunk to ready worker.
Problem with “bad” terms:



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!

$-variables I

- important tools for flow control,
- both preprocessor and algebraic objects.

They are defined during running will in general have the last assigned value, can be nondeterministic in parallel environment!

```
S x,a,b;
CF f;
L F = f(a+b) + f(a+2*b);
.sort
id f(x?$x) = f(x);
Multiply,$x;
Print;
.end
```

$-variables I

One thread could define $x and then the next thread could overwrite this value before the first thread has used it. FORM will force evaluation of such modules in sequential mode, unless the user can give FORM more information about the use of the dollar variables.

```
S x,a,b;
CF f;
L F = f(a+b) + f(a+2*b);
.sort
id f(x?$x) = f(x);
Multiply,$x;
Print;
.end
```

$-variables II

Module options for “dollar variables” concerning parallel mode:

- `minimum`
- `maximum`
- `sum`
- `local`

```
[...]
...{some code containing ‘‘dollar
   variables’’ $a run in parallel}...
moduleoption sum $a;
.sort
```

$-variables II

Module options for “dollar variables” concerning parallel mode:

- `minimum`
- `maximum`
- `sum`
- `local`

TFORM: centralized administration of the shared objects;
ParFORM: Broadcasting all $-variables in the beginning and proper collecting at the end of each module.

ParFORM is much less efficient!

RHS expressions

Right-hand side expression, e.g.:

```
S x,a,b;
L F = a+b;
L G = x + F;
Print;
.end
```

is a big problem for ParFORM since an expression may be situated in a scratch file. But different nodes may have independent scratch file systems! ParFORM forces evaluation of modules with RHS expressions in sequential mode.

For TFORM there are no any problems with RHS expressions since all threads work with the same file system.

Inparallel mode

Since TFORM has a full control on each expression it allows to implement another trick.
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>;
```

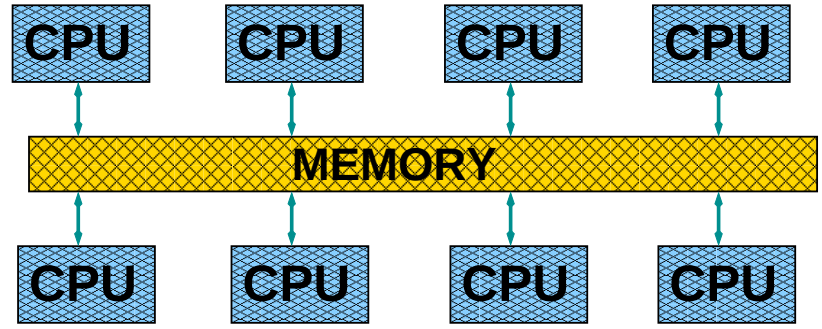
One should be careful using this statement for big expressions! Typical usage:

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

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

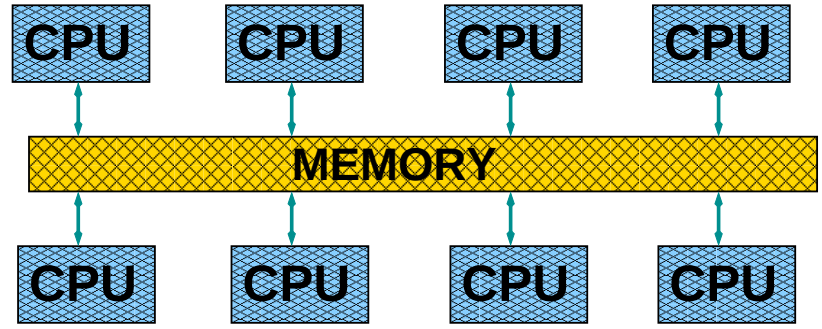
Possible hardware

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

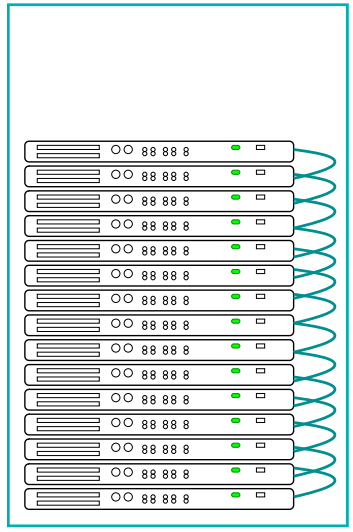


Possible hardware

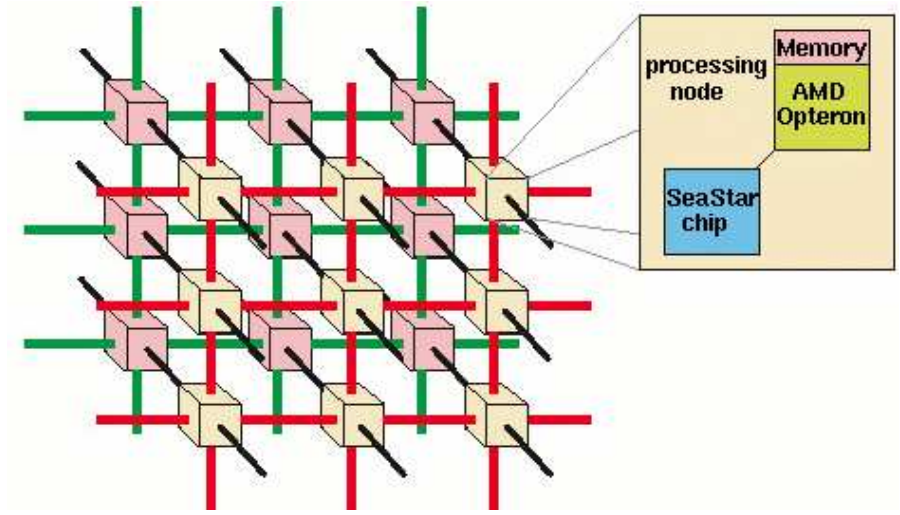
SMP (Symmetric Multi Processing), several identical processors are connected to a single shared main memory.



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

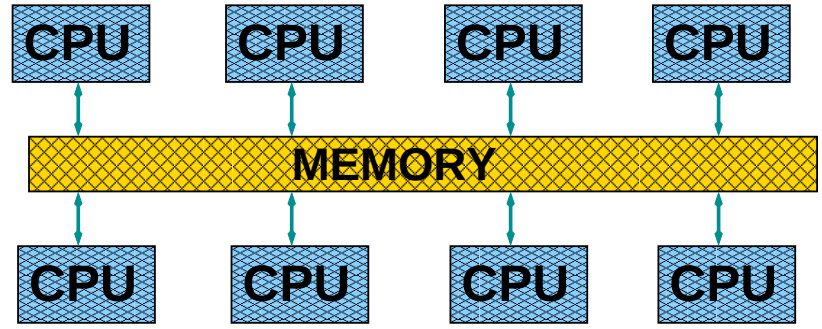


MPP (Massive Parallel Processing), e.g. CrayXT3:

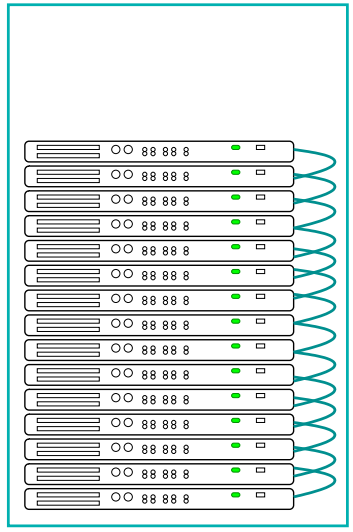


Possible hardware

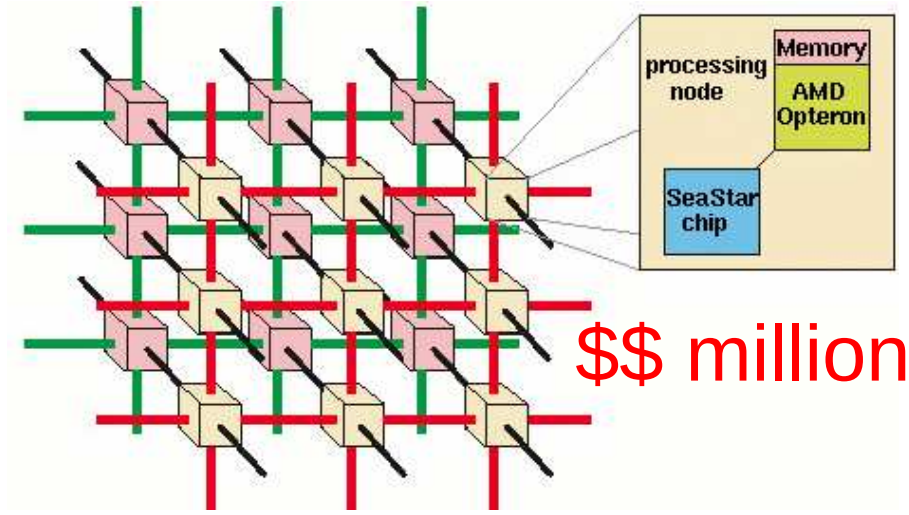
SMP (Symmetric Multi Processing), several identical processors are connected to a single shared main memory.



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



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



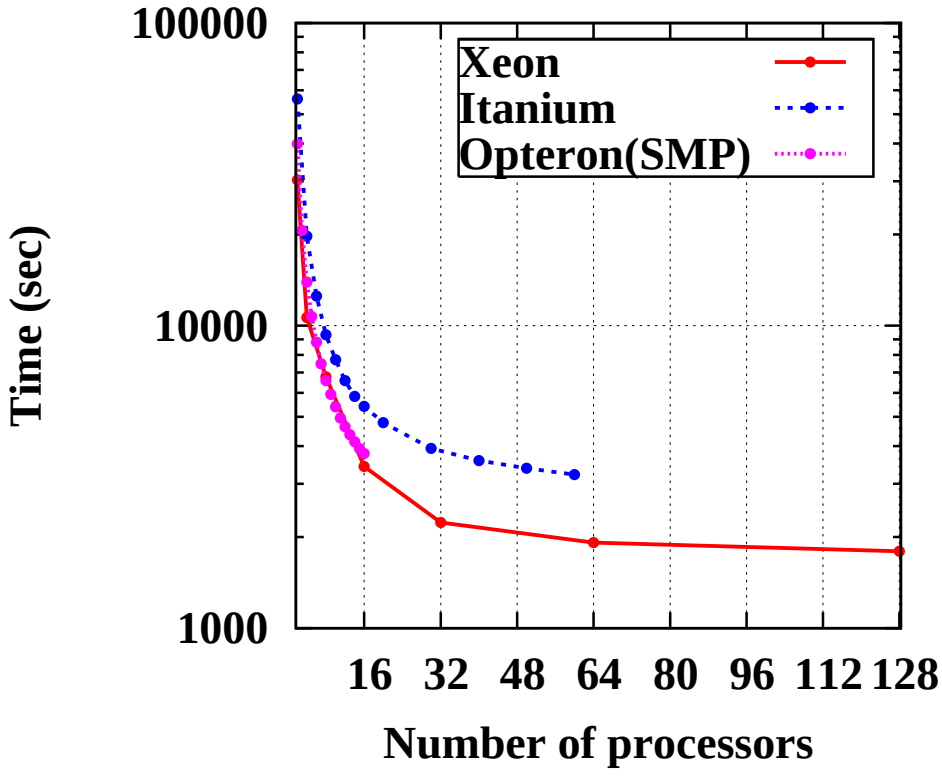
SMP vs. Cluster I

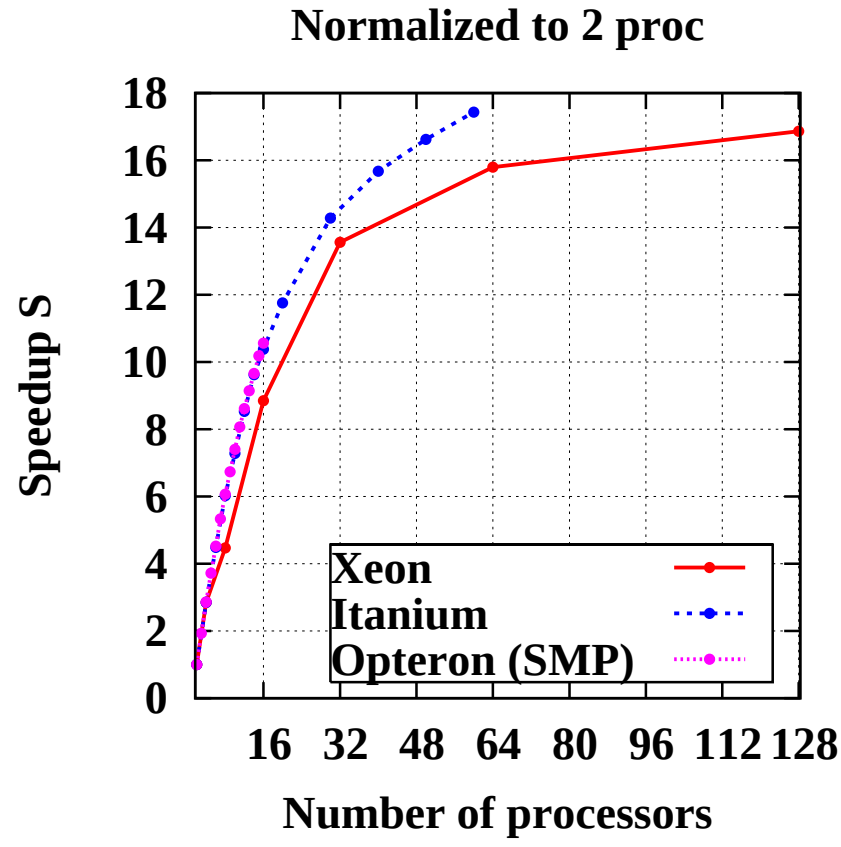
The situation changes almost every year! Retrospective:

- Year 1999: Clusters based on 10 MBit Ethernet: unusable. Clusters with Myrinet: unstable. Only SMP, non-Intel.
- Year 2003: “Cheap” clusters interconnected by Gigabit Ethernet and InfiniBand: ParFORM scales up to 4-8 nodes; “Expensive” clusters and NUMA SMP: ParFORM scales to hundreds CPUs. Our choice: Itanium based SMP computer.
- Year 2005: AMD wins, first “Multi-Core” Opteron CPUs appear. Based on dual-core Opterons, SMP NUMA-like cheap 8- and 16- core computers appear. Scalability: 2006 to 6; 2007 to 16 CPU cores!
- Year 2006: June – Intel released dual-core Woodcrest, the first Intel Core 2 microarchitecture processor, 80% boost in performance; November – quad-core Clovertown. SMP up to 16 cores, scales up to 8. Cheaper than Opterons.
- Year 2007: AMD launched quad-core “Barcelona”, completely new microarchitecture (K10), it has now L3 cache; Intel released cheap quad-core “Harpertown”, very effective and energy efficient (45 nm shrunk). Cheap clusters with hundreds CUPs available!
- Year 2008: Intel 6-core procerssors 7400-series “Dunnington”, relatively cheap SMP computers available (TFORM was tested on IBM System x3950 M2, 4x6=24 cores).

SMP vs. Cluster II

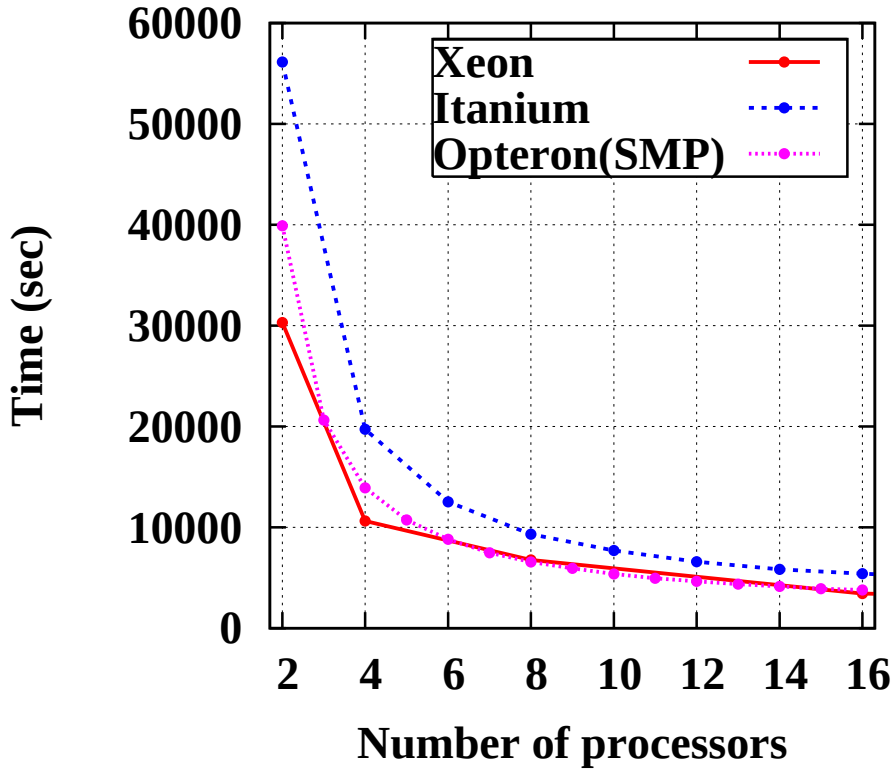
16-core SMP Opteron,
“Expensive” Itanium II Qs-NET II interconnected cluster
“Cheap” Xeon InfiniBand interconnected cluster.

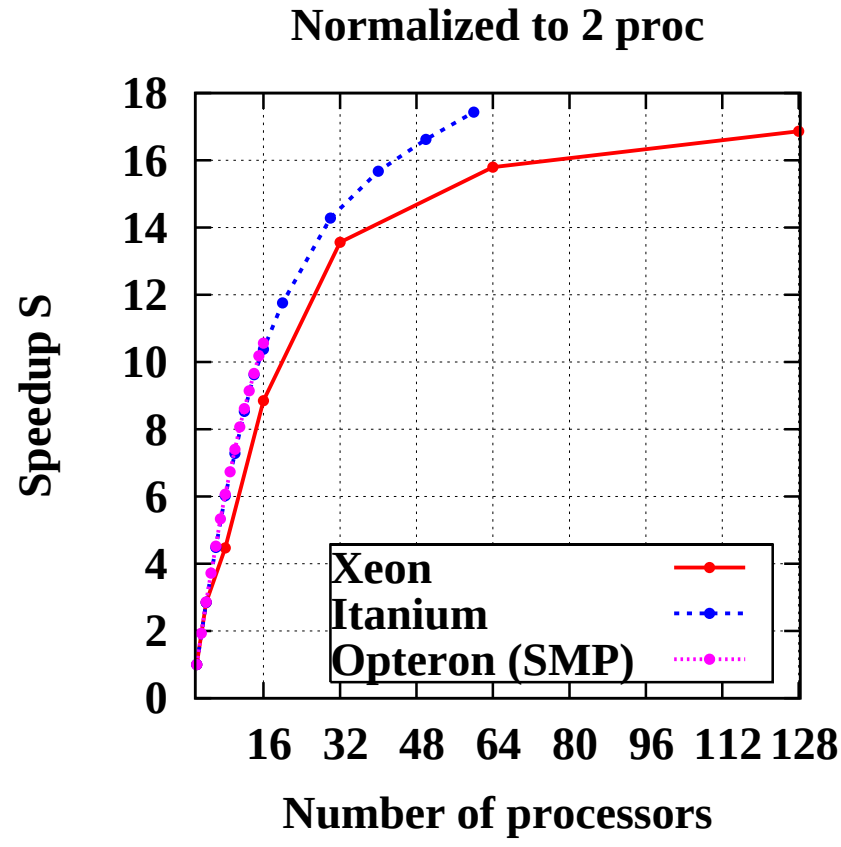




SMP vs. Cluster II

16-core SMP Opteron,
“Expensive” Itanium II Qs-NET II interconnected cluster
“Cheap” Xeon InfiniBand interconnected cluster.



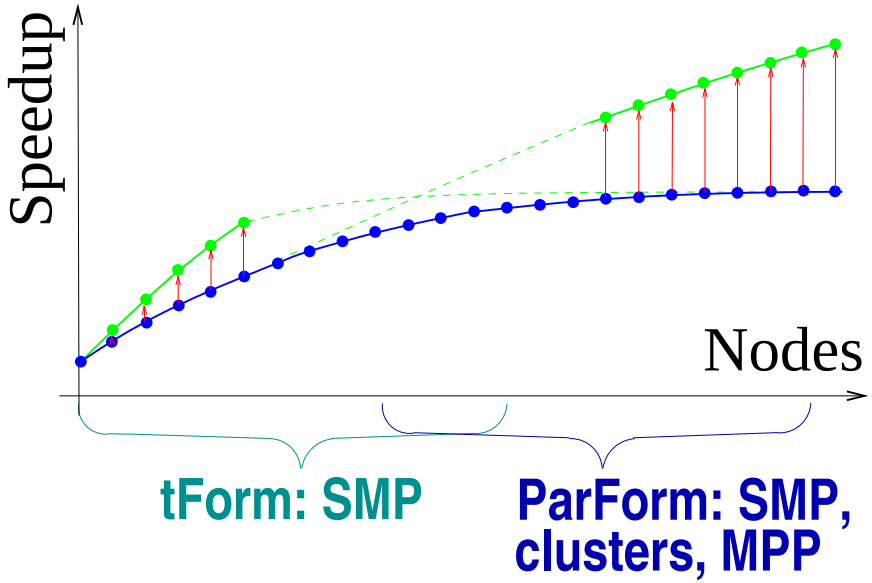


SMP vs. Cluster III

The situation may change in few months:

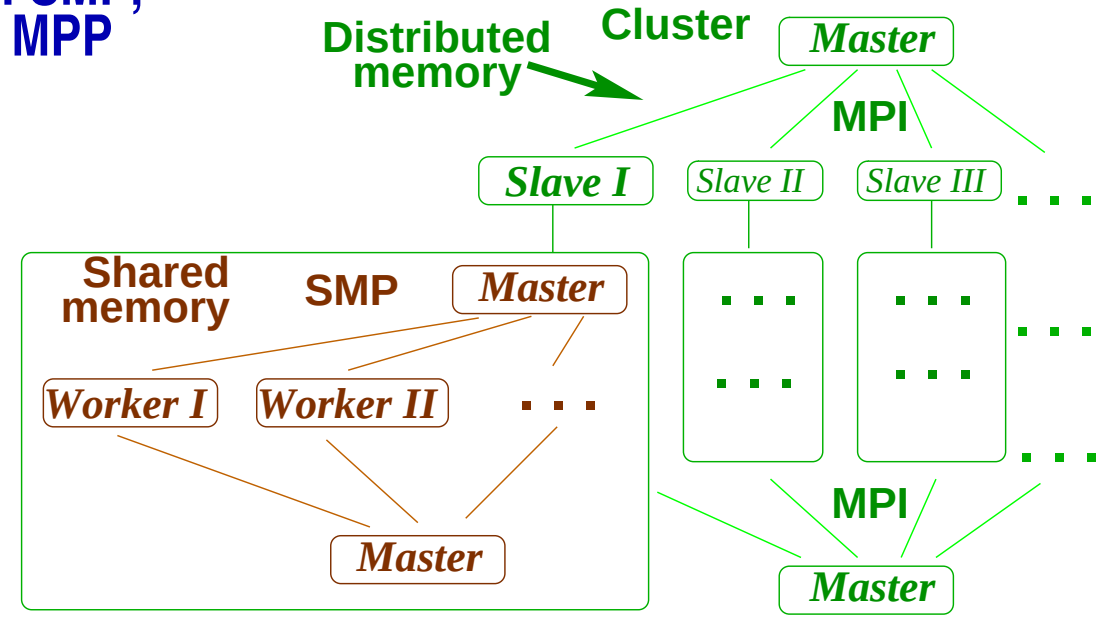
- AMD plans to launch its third-generation Opteron platform in 2009 with the “Sandtiger” octal-core processor, M-SPACE modular approach, allows to mix and match CPU features for specific tasks. Eight cores consist of eight AMD “Bulldozers” – completely new design developed from the ground up.
- Intel promised “Nehalem” (codename for both a processor microarchitecture and a processor) in 2009: 1 through 8+ cores, integrated memory controllers, “QuickPath” point-to-point processor interconnect, no Front Side Bus anymore.

Outlooks



Performance improvement;

Combinig 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.
- The present development model inspired: new features first appear in TFORM and then in ParFORM.