ParFORM: recent development

SFB-TR9, Project A2
M. Tentyukov
Fakultät für Physik, Universität Karlsruhe

FORM

J.A.M. Vermaseren
NIKHEF, Amsterdam

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

ParFORM

Fakultät für Physik, Universität Karlsruhe

ParFORM is a parallel version of FORM developed in Karlsruhe.

D. Fliegner, A. Rétey:

Working prototype 1998 – 2000.

A. Onischenko:

FORM 3.1 features support,

Large file (more than 2 Gbytes on 32-bit systems) support.

M. Tentyukov recent development

The history:

Proceedings of CASC'2004 (St. Petersburg, Russia, July 12-19, 2004), Editors: V.G. Ganzha, E.W. Mayr, E.V. Vorozhtsov.

On-line version:

http://arXiv.org/format/cs.SC/0407066

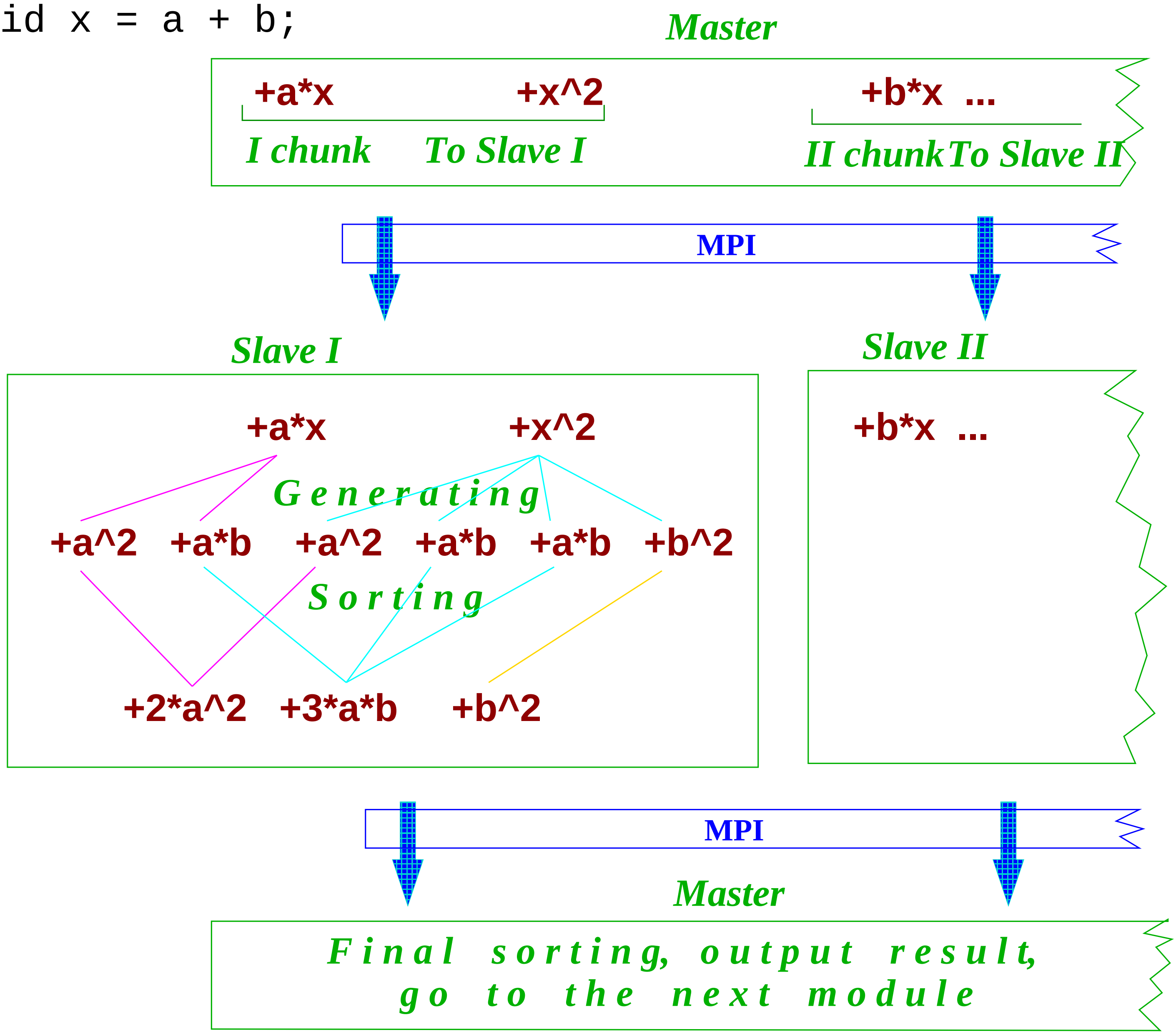
The ParFORM set-up

hep-ph/9906426, hep-ph/0007221

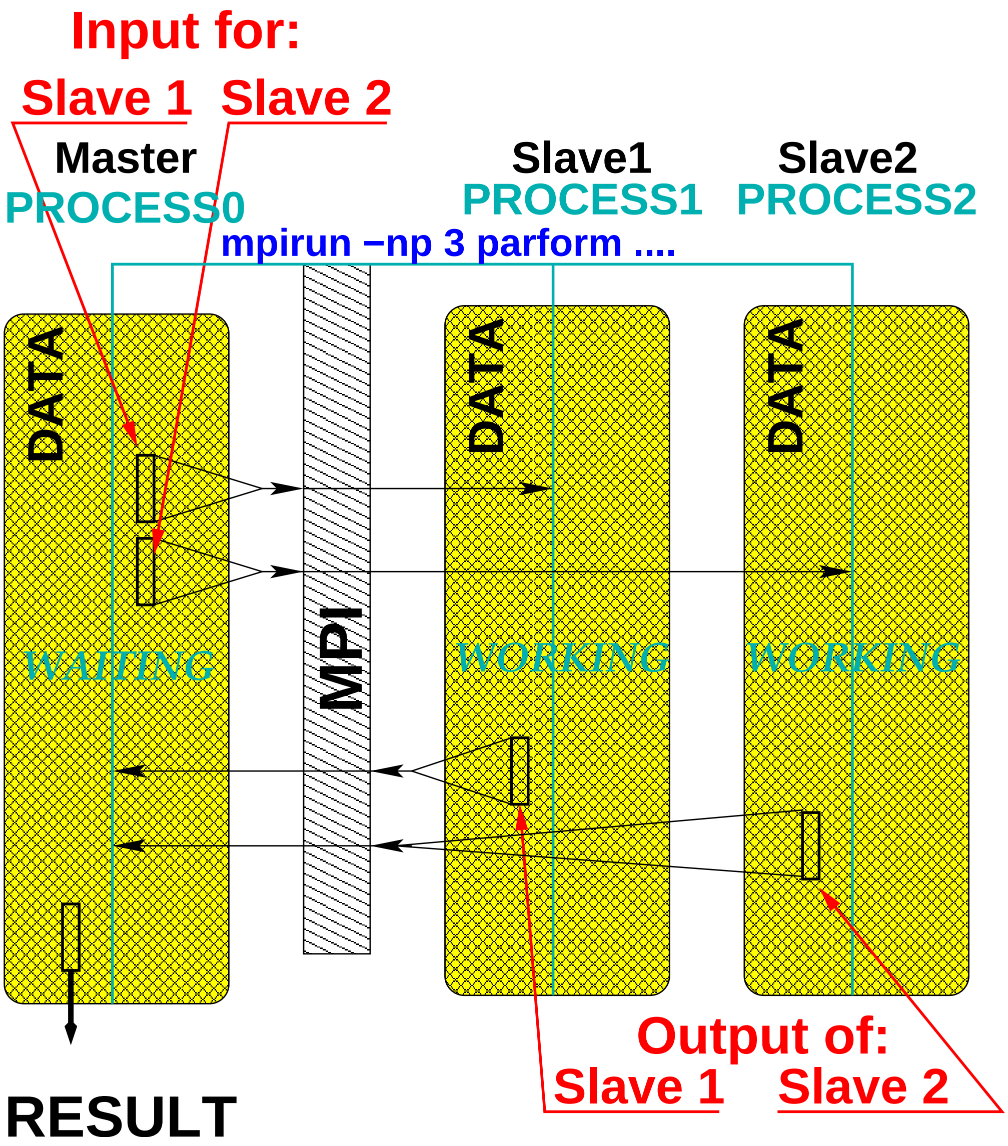
a master-slave structure.

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

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



Understanding of MPI
(Message Passing Interface)



Independent processes:

Each process is an independent stream of commands operating on independent data.

Communication:

Processes communicate by passing data as messages.

ADVANTAGES:

Universality, the program can be parallelized on one powerful computer and on cluster;

Standartization, MPI is the industrial standart.

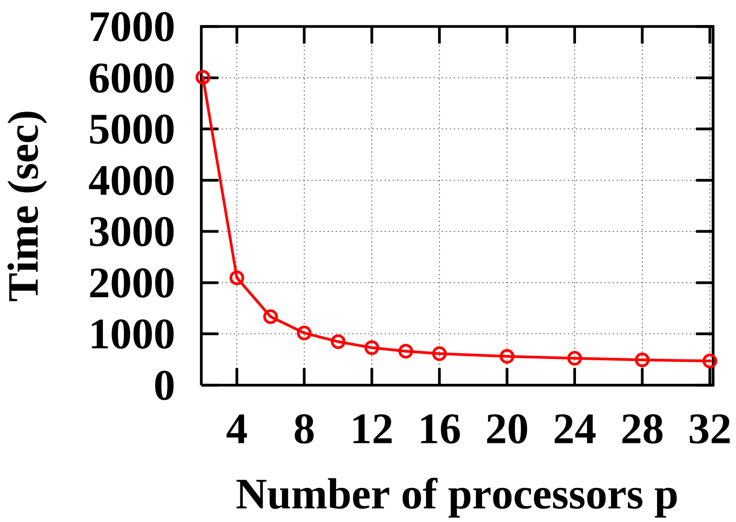
SGI Altix 3700 Server

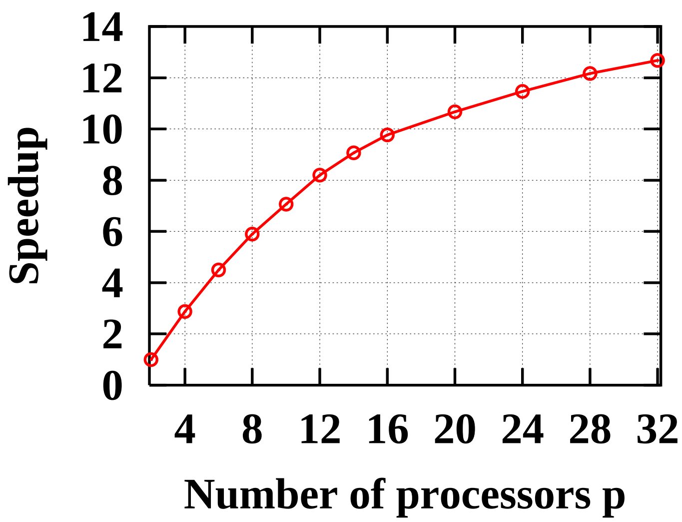
32x1.3 GHz/3Mb-SC IA64

CPUs, 64 GB DDR/166 MHz mem

2x18GB + 16x146GB SCSI disks ($\approx$ 2.1 TByte).

BAICER



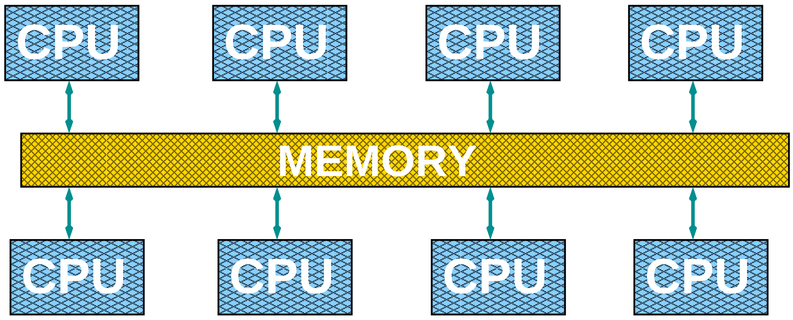


	Time (sec)	Speedup
2 processors	6009	—
16 processors	615	9.8
32 processors	474	12.6

Modern SMP architectures

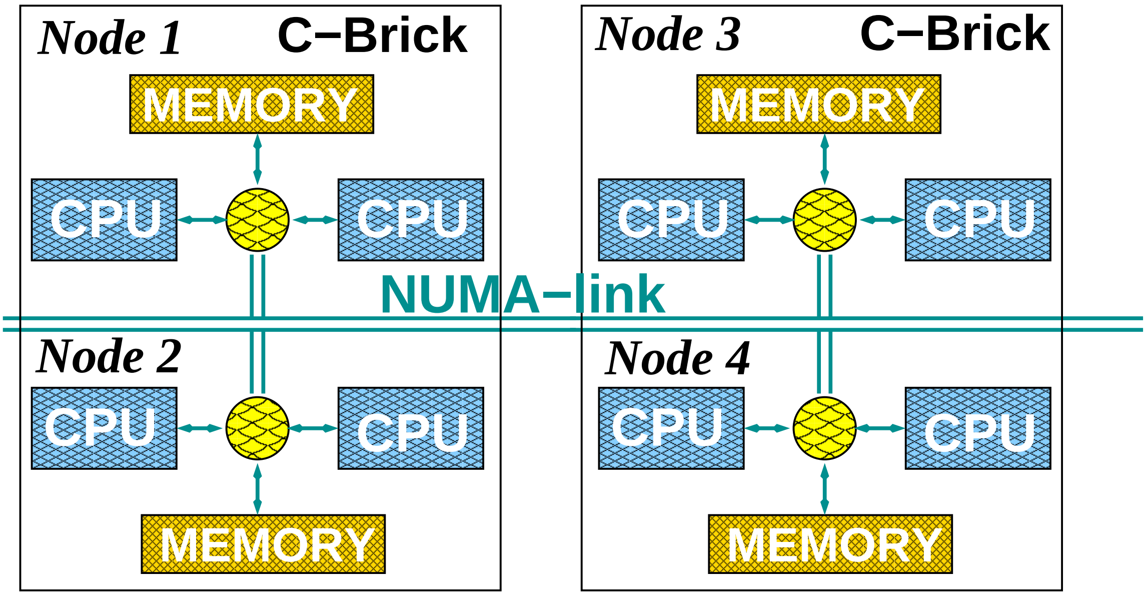
"True" SMP

(Uniform Memory Access, UMA):



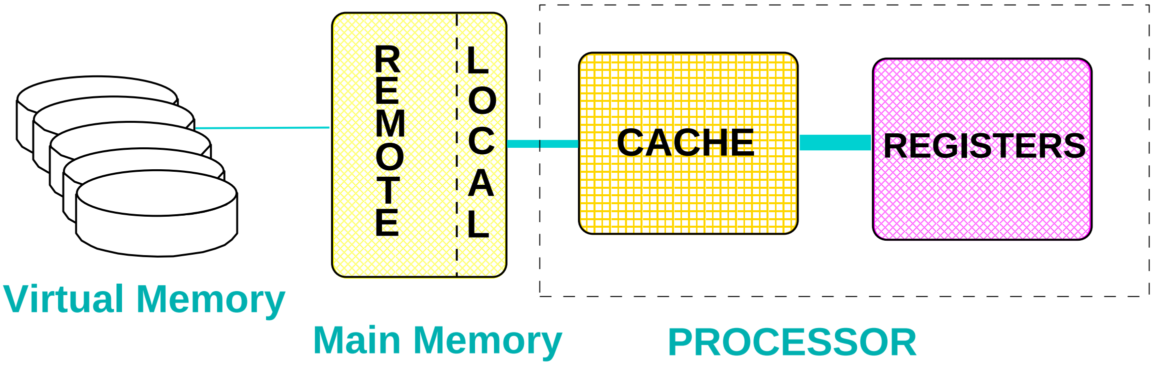
NUMA,

NonUniform Memory Access :



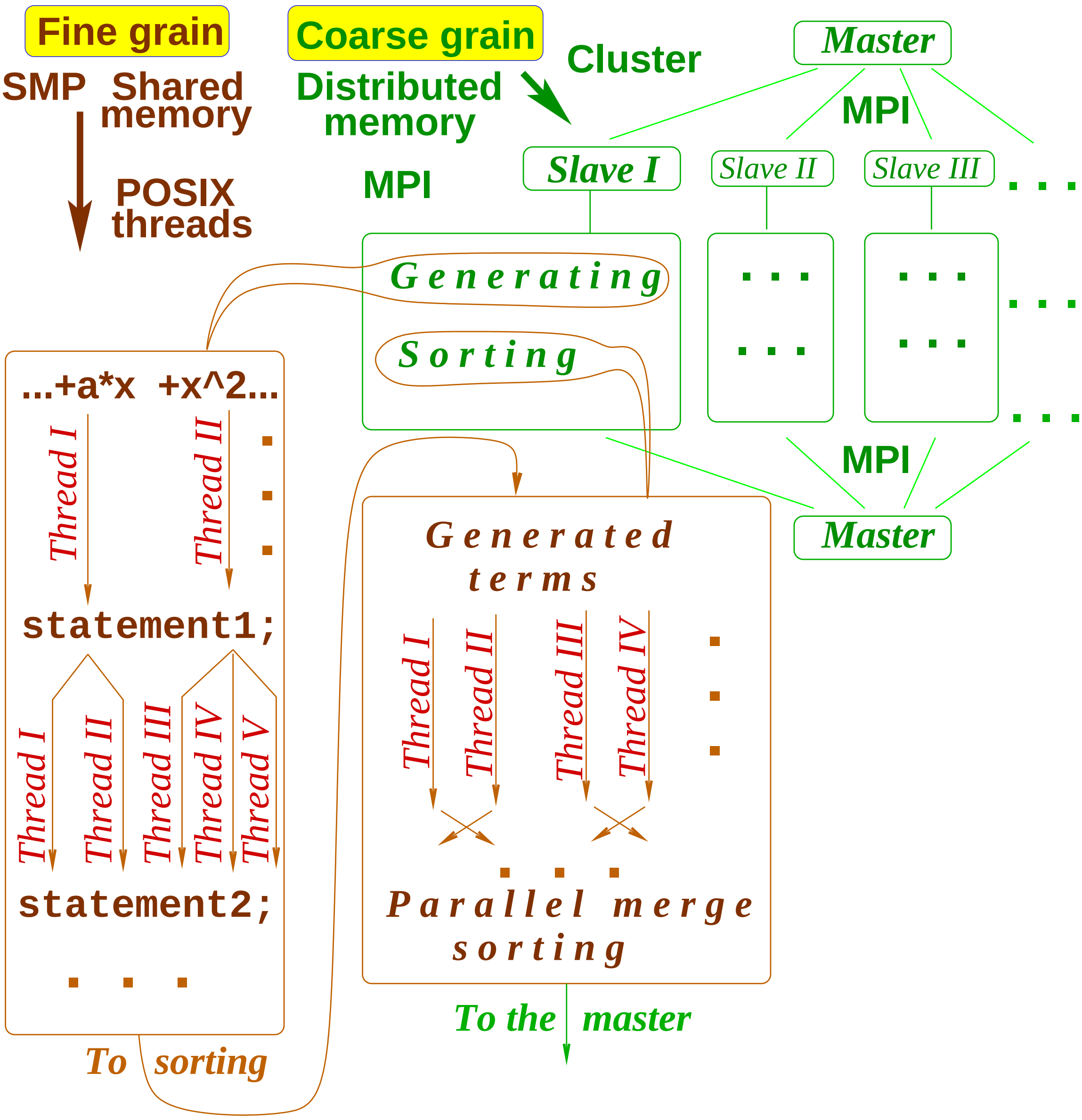
The concept of Memory Affinity: memory may be situated at different "distance". The ratio of remote to local memory access times varies from 1.9 to 3.5 depending on the relative locations of the processor.

Cache:

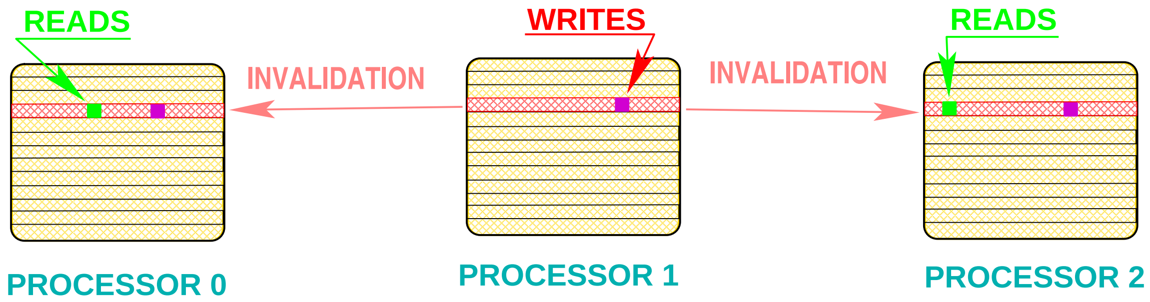


Everything is in cache! Safely assume NUMA is SMP.
Alternative to MPI on SMP:

“Fine grain” approach:



Problem with the Cache Coherency:

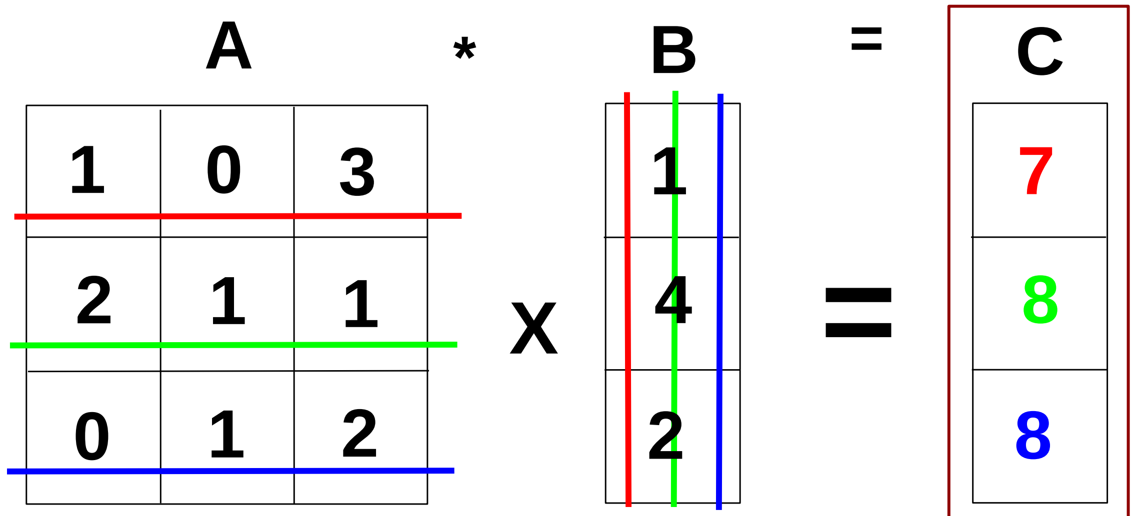


Two solutions:

- Software: slower, cheaper, better optimizable
- Hardware: faster, more expensive, less optimizable

Altix 3700: hardware solution. ccNUMA:
cache coherent Non Uniform Memory Access

Conventional multithreaded process:

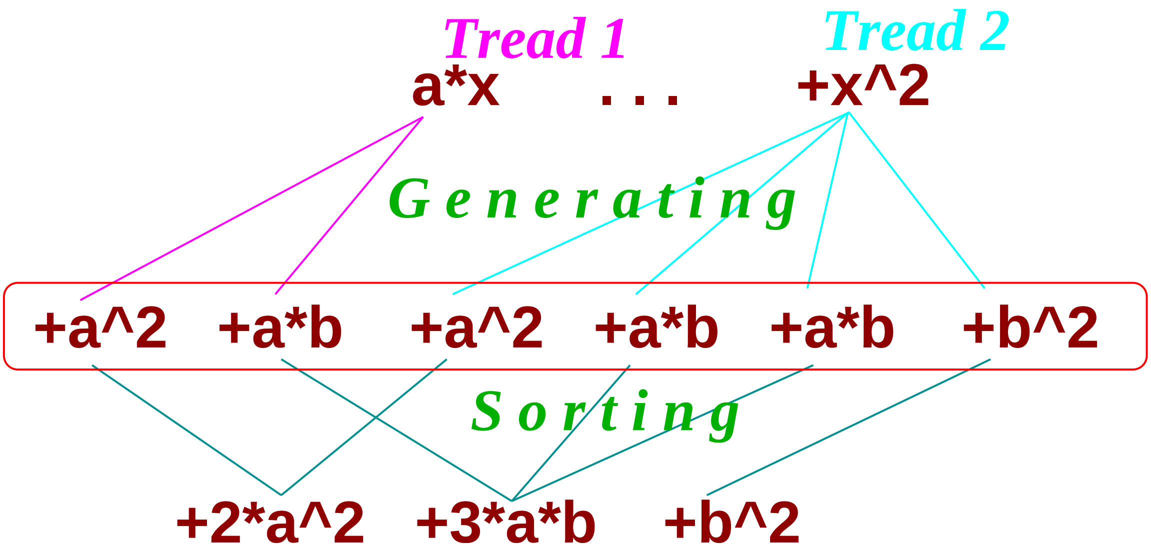


THREAD 1: L=A(1,1)*B(1)+A(1,2)*B(2)+A(1,3)*B(3); C(1)=L
THREAD 2: L=A(2,1)*B(1)+A(2,2)*B(2)+A(2,3)*B(3); C(2)=L
THREAD 3: L=A(3,1)*B(1)+A(3,2)*B(2)+A(3,3)*B(3); C(3)=L

Only reading from main memory

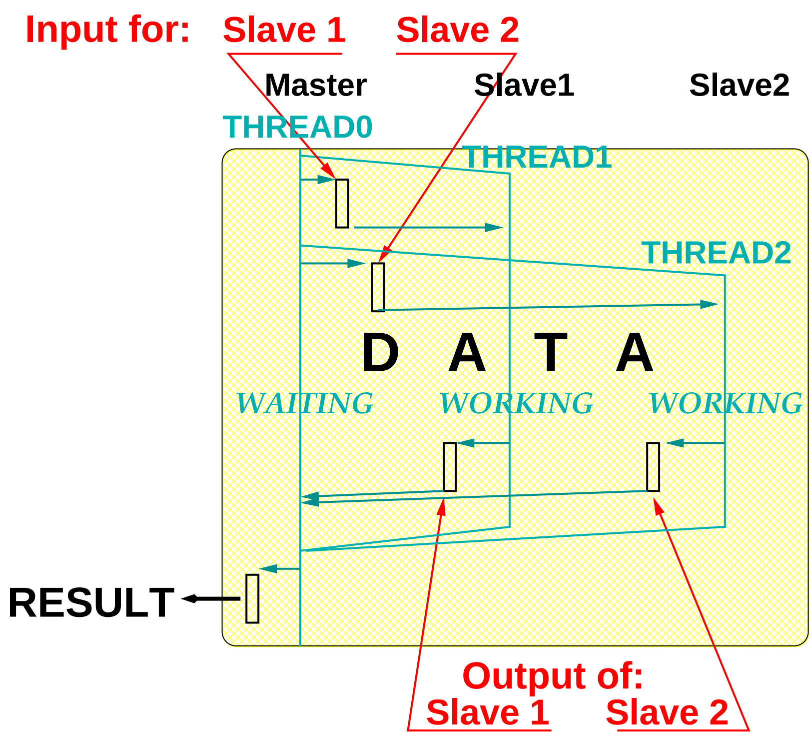
Writting to the local (register) variables, no problem with mutual cache invalidation.

ParFORM case:



Problems with "cc" on ccNUMA!

Come back to the "Coarse Grain" in order to provide data "space separation":

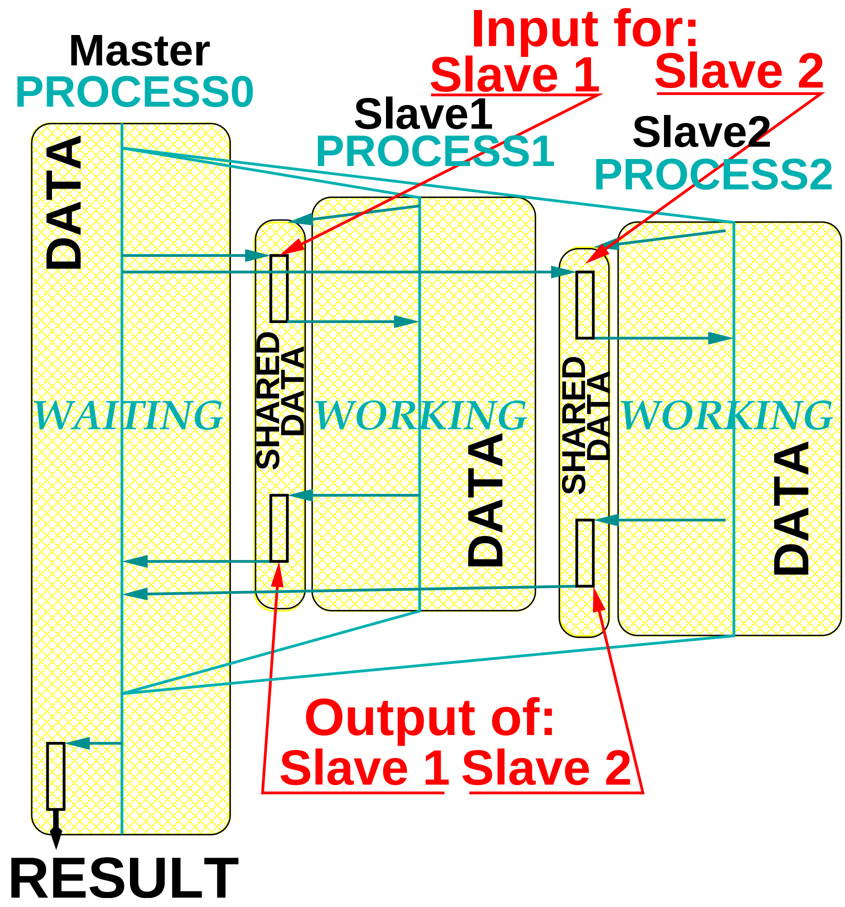


The slave thread works with memory which is affine to the processor on which the master process allocates the memory!

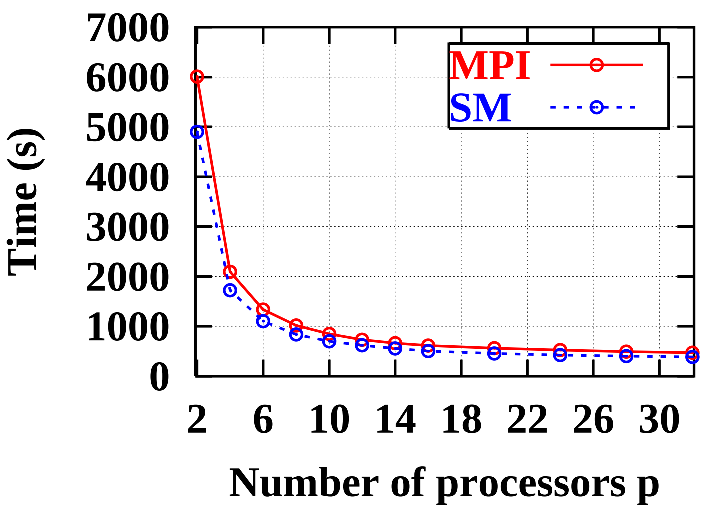
No problems with "cc", but now problems with "NUMA"!

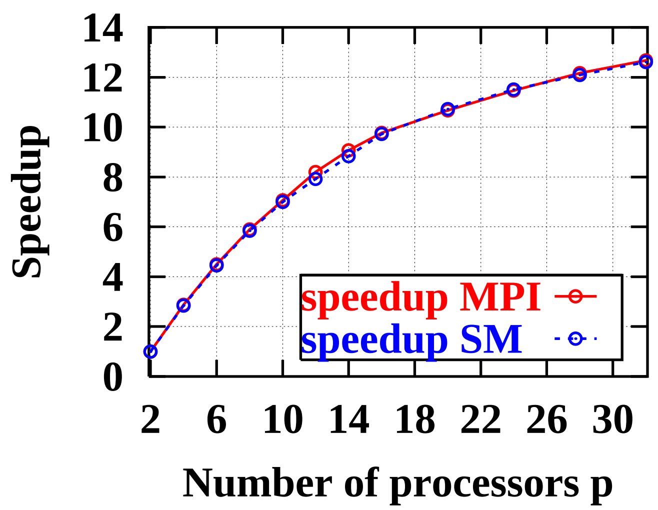
Possible solution:

allocate the shared memory buffer close to each slave:

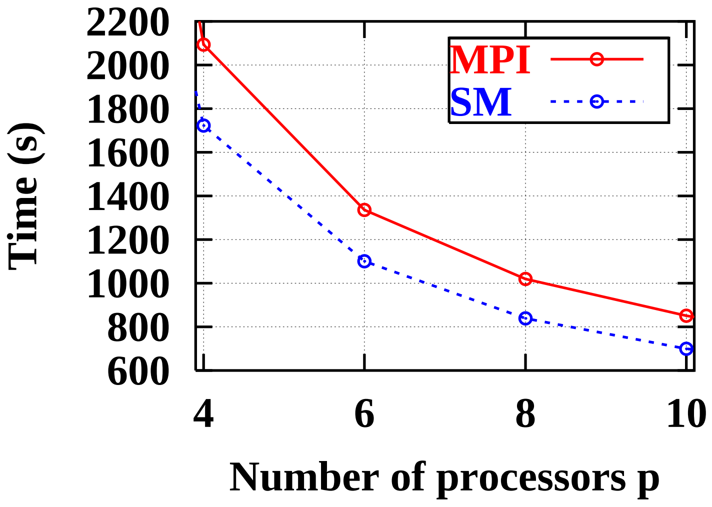


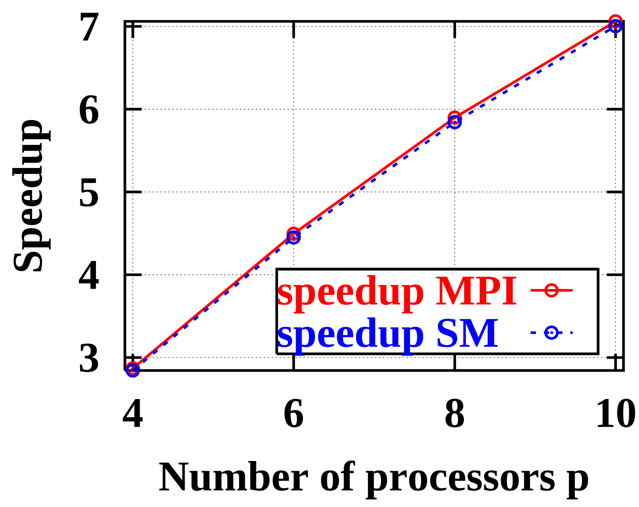
Results:





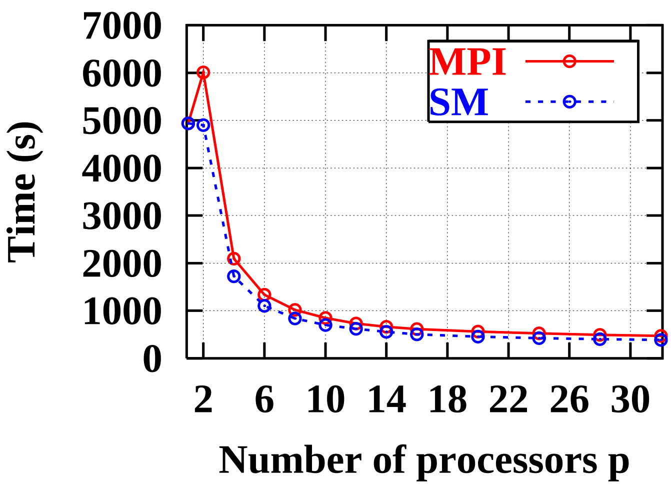
Most interesting region:

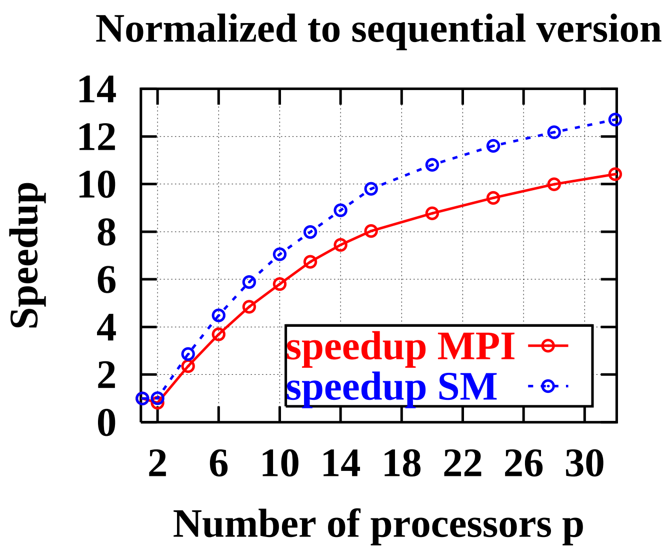




20% performance improvement!

Normalization to the sequential version:





Lesson learned:

- No communication overhead anymore!
- Bottleneck is the Master-Slave paradigm itself.

Conclusion:

- On ccNUMA computers, instead of MPI we should use the multiprocess model with multiply shared memory segments. The corresponding shared memory approach was developed, tested and demonstrates stable performance improvement around 20%.
- There is no reason to make a special multithread based version for “true” SPM systems.
- Shared memory approach permits us to manage some of known problems around non-locality like dollar variables and the `redefine` statement, and load balancing.