Parallelisation of C 4.5 as a Particular Divide and Conquer Computation

Primo Becuzzi
Massimo Coppola
Salvatore Ruggieri
Marco Vanneschi

Dipartimento di Informatica,
Università di Pisa

Target

Apply Skeleton-Structured Parallel Programming

to develop Tools for High Performance Data Mining

- Induction of Classification Trees (C 4.5)
- Deal with Huge Input Data
- In-core algorithm, out-of-core implementation through a Shared Memory, Object Oriented layer

Problem definition

- A database of k-tuples
- A distinguished attribute, the ***class*** of each tuple
- Find a useful description of the class attribute in term of the other attributes

Blue	13	47.1	Old	Metal	2
Red	12	12.4	New	Wood	1
Black	13	1.34	-	Steel	2

Tree Classification

- Build a tree of tests on the values of attributes
- Split the data set at each node

- Test attributes, using a cost function
- Select an attribute (set of attributes) A
- State the test on A, split the data accordingly
- Recursively classify the other partitions
- Nearly homogeneous partitions become leaves

C 4.5 algorithm

D1	Evaluate the information gain for each attribute, select attribute A
D2cat	**Split data into classes according to A**
D2con	**Split data in two halves, find the threshold on A**
D3	Build a new node with the test, add the homogeneous partitions as leaves
Rec	**Classify non-homogeneous partitions**
Conq	Attach the classifying sub-trees to the current node

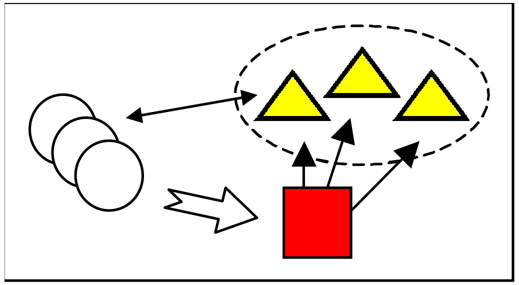
C 4.5 is Divide & Conquer

- Classification is recursively defined
- Different subtree computations are independent
- Threshold calculation is global
  - postponed, not needed until the tree is complete
  - more efficient if computed batch
- What kind of D&C is C 4.5?
  - Irregular tree structure
  - Varying computational load
  - No-cost conquer (theoretically)

Skeletons

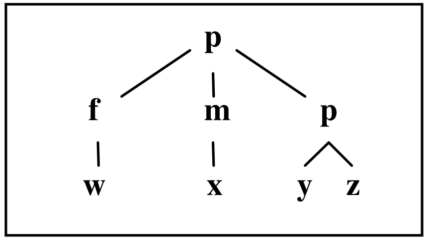
- Higher-order functionals
- Skeleton semantics are kinds of parallelism between functional modules
- Can be nested
- The program structure is described by the skeleton composition
- Implementation is independent from definition

The SkIE Programming Environment

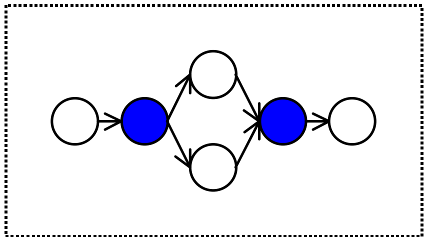


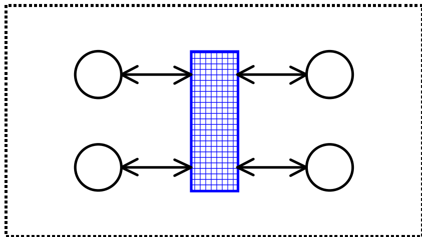
Conceptual parallelism

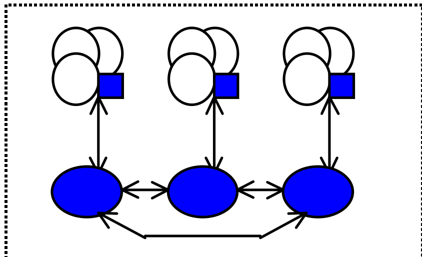
Skeleton composition



Templates **Sequential language** **+ Message Passing**	**Templates** **Sequential language** **+ Shared Memory**	**Macro Dataflow** **Dynamic** **Interpreter**







The SkIE Programming Environment

- Structured parallel environment
  - Coordination language
- High-level view of parallelism
- Programmability, Portability, Performance
  - Code and performance portability
- Fully or semi-automatic optimization of low-level code using profiling and performance models.

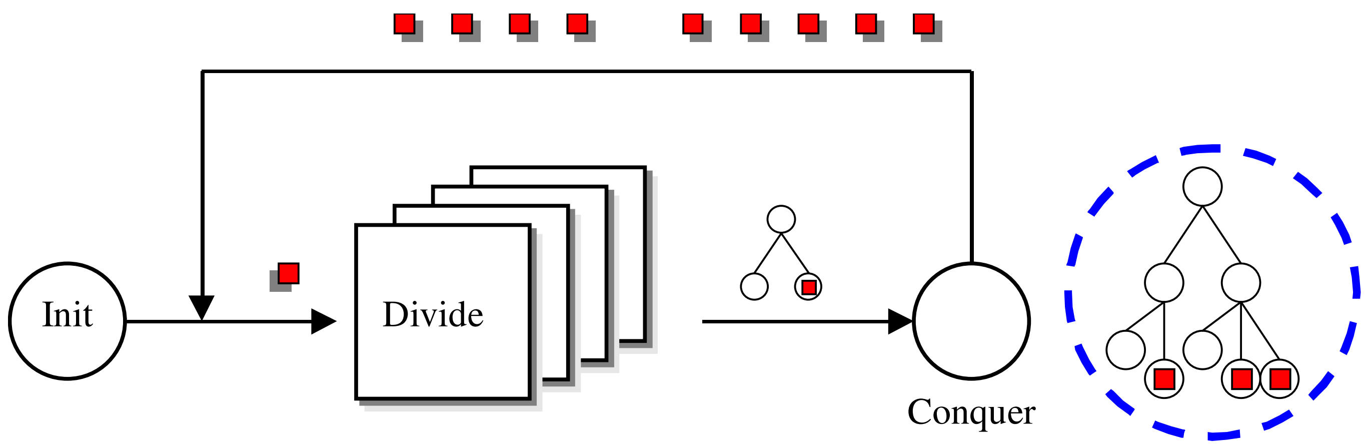
Separation between the application programmer's and system programmer's views of parallelisation

Parallelisations of C 4.5

- Data parallelisation (row distribution)
  Distribute the tuples over the processors
  Synchronous, poor if there are many small nodes
- Attribute parallelisation (column distribution)
  Each processor holds an attribute
  Limited parallelism, more problems of load balancing
- Task parallelisation (node distribution)
  Each recursion becomes an independent computation
  Ineffective for the large nodes that stem from the root

Pure task parallelisation

- Task-parallel skeletons hide details about the message passing support
- Dynamic load balancing using standard on-demand scheduling



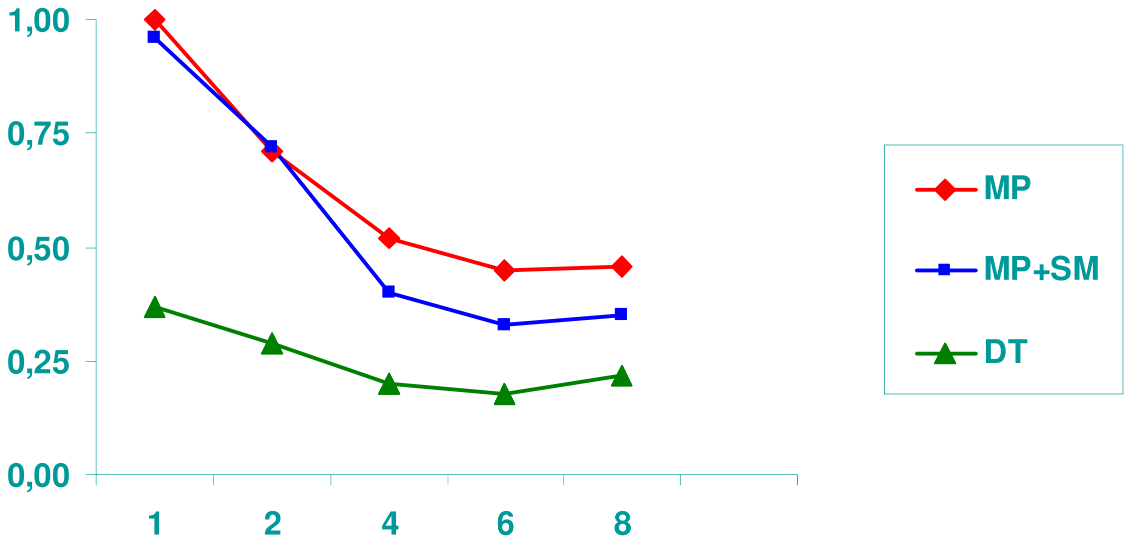
In-core and out-of-core

- Kinds of operations on the data
  - sort, scan, search, partition
- Unknown and sparse access patterns
- Heavy overhead if data don't fit in memory
- Sorting as an example
  - O(N log N) disk/remote operations worst case
  - O(n $\log_m$ n) with secondary memory algorithms where n=N/B and ($\log_m$ n) is bounded in practice.

Development path

- Start with a task parallelisation
- Add sharing facilities
- Move to a fully D&C computation
- Object-Oriented support for sharing data
  Distribute large data across the aggregate memory
- ➡ Add collective operations
  Mix data and task parallelism
- ➡ Exploit external-memory algorithms
  to optimise the collective operations

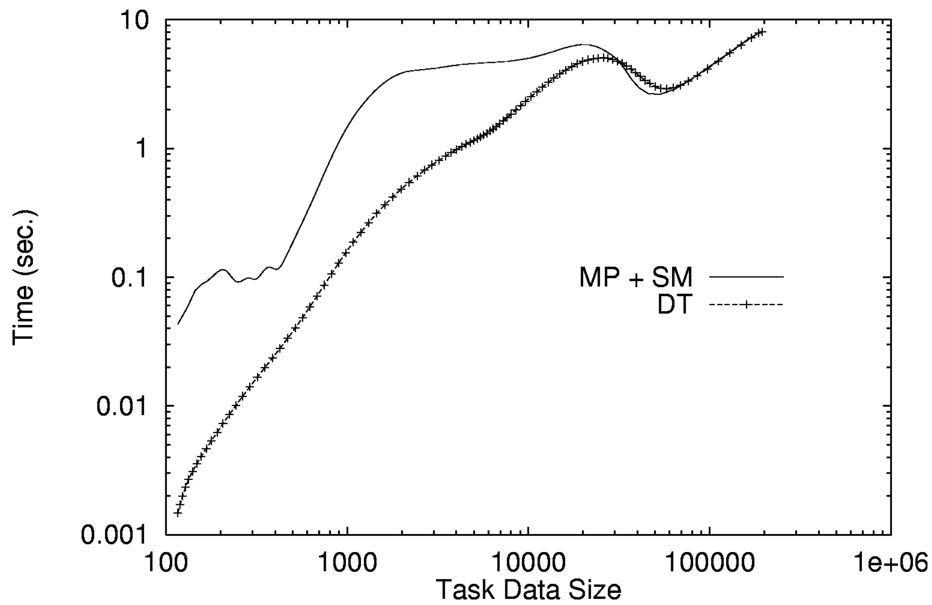
Sharing versus Communication



- MP + SM : First experiment with shared data
- DT : Threshold is delayed until the tree is complete

Fully D&C strategy

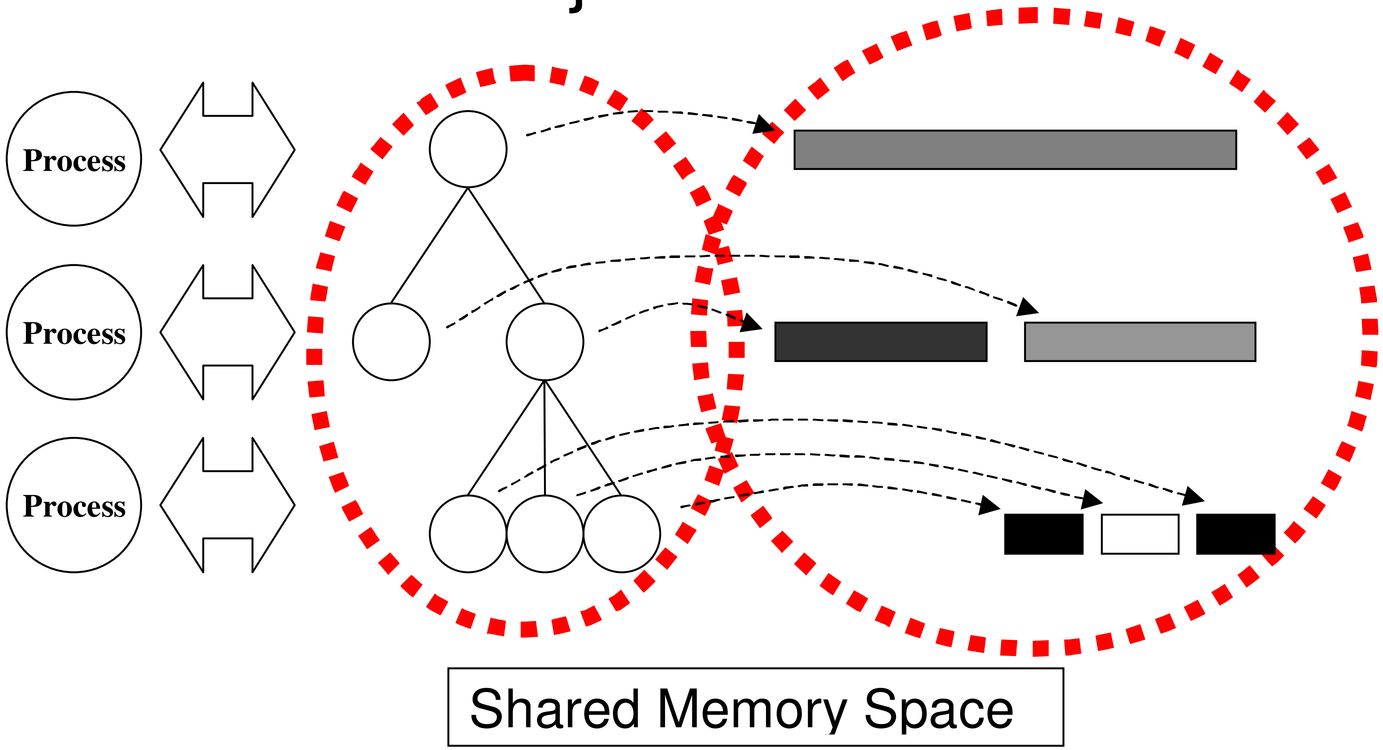
- Computational load per task is
  - Lower
  - More clearly related to task “size”



- Ordering the task by size improves load balancing (expand bigger tasks first)
- A simple priority queue in the Conquer module

Ongoing work: global tree & data

- The input data and the tree structure become shared remote objects



Summing up

- The parallelisation of large and irregular problems benefits from exploiting the sum of the local memories
- Shared memory space = lower memory layer
  - Carefully design data structures and algorithms to fill in the speed and size gaps
- The object abstraction can hide these details
  - Implementation of the access methods doesn't clutter the application algorithm

Future work: Remote objects and operations

- Application tuning
  - The data are shared and allowed to grow in size
  - Removal of the centralization point in the Conquer phase
- Better exploitation of data parallelism
  - Collective operations on shared partitions (scan, partition, sort)
  - Could make use of external memory algorithms to efficiently access the shared address space