A model for performance evaluation of message passing architectures in spatial data processing(*)

Enrico Nardelli(1,2), Carmine D'Amico(1), Marco Santacroce(1)

(1) Dipartimento di Matematica Pura ed Applicata, Univ. di L'Aquila, Via Vetoio, Loc. Coppito, 67100 L'Aquila, Italy, e-mail: nardelli@vxscaq.aquila.infn.it.
(2) Istituto di Analisi dei Sistemi ed Informatica, C.N.R., Viale Manzoni 30, 00185 Roma, Italy.

Abstract

In this paper we give a first formulation of a model for evaluating performances of a message passing architecture parallel machine in the context of spatial data processing. We consider 2-dimensional data of the type 'region' and analyze operations of union and intersection between them. On the basis of the characteristics of manipulation algorithms and of the architecture we individuate as the best way of implementing them, we propose and validate through experiments a model able to estimate time required to execute union or intersection operations between two regions of arbitrary shape as a function of a small number of parameters describing input data. Though derived on a specific machine, the model obtained is of general validity, since only the values of numerical constants are dependant by the machine used. Such a model is the first necessary step in tackling the issue of query optimization for spatial data in a parallel environment. The work is a part of a more general research programme aiming at studying the best approach to take advantage from parallel architectures for spatial data processing.

1. Introduction

Efficient processing of spatial data is of increasing and increasing importance. It is a basic step in many advanced application environments, such as CAD, image procesing, scientific visualization, environment monitoring, land planning, and so on.
But to efficiently processing spatial data a high computing power is required, since applications usually deal with very large amounts of data and manipulations required may be very heavy (complex transformations repeatedly applied many times) [Arc92, Arc93b].

An help may come from the use of parallel architectures, since computation on 2-dimensional data sets may be easily partitioned and assigned to different processing units working in parallel. This issue is examined in the ITU-LAND project, in the framework of the ESPRIT's research programs, where the use of high performance architecture for land planning is considered [ITU92].

In this work we consider 2-dimensional spatial data of the type 'region', which is one of the most important classes of such data in the context of geographical data management. Basic operations used for region manipulation are union and intersection, since others may be derived from them.
Regions can be represented using a 'vectorial' approach, that is maintaining for each region the list of segments which make up its boundary. This approach allows a 'high fidelity' representation but makes union and intersection operations quite complex since computational geometry algorithms are required. To an opposite side there is a 'low fidelity' representation which maps each region to its bounding box. In this case operations may be performed with high efficiency and using highly performant data structure, such as the grid-file [Nie84], but the loss in fidelity may be unaffordable. An itermediate way is the so-called 'raster' approach where each region is represented by the set of points of a discrete decomposition of the 2-dimensional space which are covered by it. The loss in fidelity is here limited by the discretization step and union and intersection operation may be efficiently implemented since are operations on set. When this approach is followed, a widely used data structures, which allow to save space where regions are more regular, is the 'refion quadtree' [Sam90].

Previous work has been done by Chien and Kanada [Chi90] with reference to a distributed framework, focusing on how to split computation and balancing load, but only a static schema of allocation of work to processors plus algorithms to transfer work if some processor becomes idle is proposed and no general solution is given. Algorithms for parallel processing of spatial data using quadtrees have been studied and analysed in a general and abstract way by Bestul [Bes92].

With regard to the bounding-box based approach, work has been done in a parallel environment by Kamel and Faloutsos [Kam92] using the R-tree. But, as remembered above, R-trees do not allow to exactly represent shapes of region, since they approximate a region with its rectangular bounding box.
Here we follow the approach of Bestul, but considering the impact of real parallel architectures on an environment for

(*) Partially supported by project "Multidata" of the CNR's research program "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo" and by projects "Amusing" and "Ituland" of the ESPRIT's research program "Basic Research Action".

spatial data processing.

The structure of the work is the following. In section 2 we revise data structures and algorithms used for processing spatial data of the type region in a parallel environment. In section 3 we describe the characteristics of the environment used for experiments in terms of machine, data, and system software. In section 4 we analyse the most important issues from the software architecture point of view of the experiments and how we tackled them. In section 5 we define the model and identifies its parameters. Section 6 contains conclusions and directions for future work.

2. Spatial data structures and algorithms

In this section we briefly describe data structures and algorithms used for parallel quadtree processing.

Spatial data we consider are described by sets of points belonging to a raster decomposition of the plane. That is, we represent the space $\Re^2$ with its discrete approximation Z^2. This is not a reductive assumption for representing the spatial/geometric nature of geographical entities, given the limitations of the finite representation of real numbers available in computers, and is considered an important aspect of a logical data model for spatial data [Güt92].
Thus, a region is a finite subset of the points of the plane that have integer coordinates. No assumptions on its connectivity status is made. We assume the universe of discourse is a finite subset of Z^2. The fundamental operators for any sound and complete manipulations of spatial data are union, intersection and negation [Gar91a, Gar91b]. Notice that, since we are working with a finite universe set, negation may be safely used.

A widely used and long-time studied approach to the representation of regions defined as above is the quadtree [Sam90] The term quadtree is generally used to denote a hierarchical data structure developed on the basis of a regular decomposition of the space. The hierarchical decomposition is data-driven, but always proceeds according to a regular scheme, going to deeper levels only where represented features are more densely distributed. In this way space is saved where the distribution is more scarce.

Assuming to have at disposal a binary image of size *T*x*T* (e.g., pixel elements), where *T* is such that there exists an integer such that 2^m=*T*, we proceed in the following way: at height 0 there is the whole image, of side length *T*. At the first stage of decomposition the image consists of four quadrants of side length *T*/2. At a second stage each quadrant is then subdivided into four quadrants of side length $T/2^2$ and so on. The decomposition stops either when a quadrant is wholly covered (it is said to be *black*) or wholly uncovered (it is said to be *white*). We shall use also the term *block* to denote a quadrant. The decomposition can go on until the pixel level, with quadrants of side length $T/2^m$. The decomposition can be represented as a tree of outdegree 4, with the root (at height 0) corresponding to the whole image and each node (at height *d*) to a quadrant of side length $T/2^d$ The sons of a node are, in preorder, labeled NW, NE, SW and SE. For a given image, nodes are then black, white (leaf nodes) or grey (intermediate nodes). Correspondingly, we speak of black, white and grey blocks. Look at the figure below for an example:

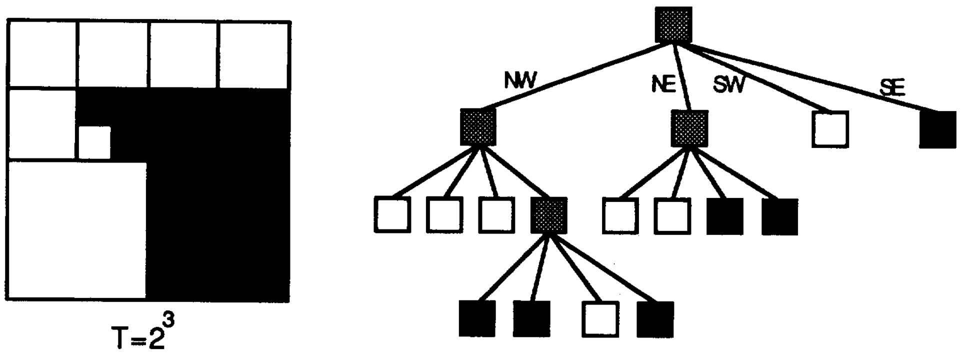

Figure 1: A binary image and its quadtree

The way quadtree is defined leads naturally to a pointer-based representation. But when dealing with large quantities of spatial data a pointer-less representation is preferrable since it is more space efficient, improves performance of sequential operations, and is better suited to a disk based management of data. A widely used pointer-less approach is the "collection of leaf nodes", also called the *linear quadtree* [Gar82, Abe83, Nar93a]. See

[Sam90] for other representations.

With such an approach we only represent the leaf nodes of the quadtree, associating to each of them:

(1) a *locational key* (*l-key* in the following), corresponding to a sequence of digits describing simultaneously the path from the root to the node and the depth level;

(2) a *colour bit* which is set to 0 if the leaf is white and to 1 if it is black.

The l-key for a node of height d in a 2^mx2^m space is recursively defined as follows. Define the l-key for the root as an all-zero string of length m. Let the immediate ancestor of k have the key k'. Then:

$$k=k'+s5^{m-d}$$

where:

s=1	if k is the NW son of k'
s=2	if k is the NE son of k'
s=3	if k is the SW son of k'
s=4	if k is the SE son of k'.

The l-key is then a base 5 code with m as fixed length. So, for example, the l-keys over a 2^2x2^2 space are the following:

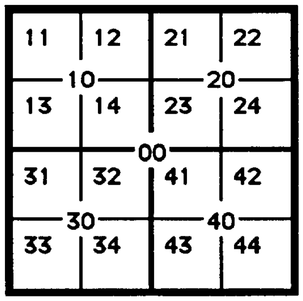

Figure 2: Locational keys in a 4x4 space

If the list is kept sorted according to ascending values of the l-keys we obtain a total ordering of the 2-dimensional space. Such an order is called also Morton order or Z-order [Sam90]. As an example, for the quadtree in figure 1, we have the following sequence:

(110,0),(120,0),(130,0),(141,1),(142,1),(143,0),(144,1),
(210,0),(220,0),(230,1),(240,1),(300,0),(400,1).

Algorithms for union and intersection between two quadtrees are very simple to define, since they correspond to a synchronized visit of both structures. Synchronized means that at each step the algorithm is visiting a couple of nodes, one for each tree, which are at the same depth and have the same label. At each step, when confronting these corresponding nodes, the action to be taken is decided according to a truth table. Here below, in figure 3, truth tables for both operations are reported, where either the resulting color of the operation or what it needs to be done is indicated.

As you can see, if both nodes are leaves then the result can always be immediately computed. Otherwise, different actions have to be taken for the two operations. More precisely, 'recurse' means to recursively visit all sons in both structures in synchronized way, 'subtree' means to return the whole subtree of the grey node involved in the operation.Both algorithms can be easily trasformed to work on linear quadtrees.

Due to their recursive nature which divide the space into disjoint subregions, these algorithms can be made parallel quite naturally. See [Bes92] where a complete logical framework, including a parallel computation model, has been proposed for quadtree manipulation. Here we can just recall the basic idea. Suppose we are given a certain number of logical processing units. Then, instead of recursively visiting the sons of the current nodes, as the sequential algorithm does, the parallel algorithm invokes four new logical processing units, transmit them the four subregions to examine, and waits for the result. Each logical processing unit then executes the same algorithm: if it is able to compute the result then returns it to the calling unit otherwise calls four new logical processing units. When all leaves have been visited the algorithm terminates and the logical processing unit which is the root of the whole process has the result.

UNION	White	Black	Grey
White	White	Black	subtree
Black	Black	Black	Black
Grey	subtree	Black	recurse

INTER.	White	Black	Grey
White	White	White	White
Black	White	Black	subtree
Grey	White	subtree	recurse

Figure 3: Truth tables for union and intersection

3. Experimentation environment

The approach described in [Bes92 for quadtree processing in a parallel environment gives a unified a coherent logical view of the whole subject, but needs to be expanded and refined when considering real architectures. We have analysed message passing architectures and studied how the architecture affects time performances.

In this section we describe the environment used in our analysis.

3.1 Machine

The machine we have used is a MEIKO Computing

Surface with five boards having four 32-bit processors each. There is no shared memory among processors. Each physical processor has about 1.9 Mbytes of private RAM, and can host up to 59 logical processes (i.e. software modules or *tasks*), but without support for virtual memory management. On each board an advanced interface unit is able to guarantee reconfigurable physical communication links between processors. A physical communication link can be established between any two of the processor, and each processor can have at most four of them.

The machine is therefore able to support any communication network whatsoever. It needs only to declare which is the logical topology of the needed communication network. At compile time such a logical topology is realized using the physical links, and if these are not enough (e.g. when five links are requested for a given processor), additional routines from the communication software[1] of the parallel operating system are automatically loaded so that at run-time the required logical topology is available in a transparent way to the user.

Allocation of logical tasks to physical processors is done at compile time of the parallel program, tasking also into account the declared topology of communication, and is fixed until the termination of the parallel program.
MEIKO machine interacts with end-users through a SUN/SPARC executing interface as well as disk management functions. Secondary memory available to parallel processors is therefore only what is offered on the SUN/SPARC and is a serialization point for a parallel program.

3.2 Data

Data used for experiments are 512×512 binary images represented with linear quadtrees. Notice that the choice of using linear quadtrees, that is list of nodes, is forced by the parallel paradigm used. In fact, to work with pointers is extremely awkward in the message passing context since in this case data need to be transformed back and forth each time they have to be passed from a processing unit to another, which uses its address space in a different way. Due to the absence of virtual memory management we have not been able to experiments with larger images.

Images we have used have been either produced manually from geographical maps, or randomly generated. The random process used to produce data is a branching process where, starting from the root (at level 0), each node of level k has a probability $\frac{1}{2(k+1)}$ to be grey; if it is not grey then it is black or white with the same probability. This approach gives raise to random images that are more similar to geographical maps than other approaches. We have no space here to discuss probabilistic models for spatial data (see [Nar93b] for a complete discussion of the topic)

3.3 Software environment

The system software environment available in the MEIKO machine, though supporting all basic functionalities needed in a message passing architecture, offer no help in implementing the logical framework describe in [Bes92].
The main point is that there is no practical way of dynamically creating tasks (which is the corner stone of the Bestul approach). In fact, we made many different attempts, but the only possibilities were of so limited functionalities ot be of no practical use. We found that:

- a task which dynamically creates other tasks may be run only on the interface SUN/SPARC and not on the parallel processors: it is therefore impossible for tasks running on one of the parallel processors to dynamically create new tasks;
- when a task dynamically creates new tasks it has to suspend its activities until all spawned tasks have terminated their work: it is therefore impossible for it to exchange messages with spawned tasks; the only thing it can do is to create all offsprings it needs and to wait for the termination of all of them;
- tasks which have been dynamically created can communicate only among themselves and cannot exchange messages with statically created tasks;
- dynamic creation of tasks may be done only once in the life of a task.

Given the above constraints, the only viable solutions is the static creation at compile time of a number of tasks not less than the maximum number of tasks needed and to dynamically activate/deactivate them according to current needs. The management of static tasks which are dynamically activated and deactivated now requires an additional task which takes care of the overall management of the parallel program.

Now the main question is: how many tasks to create for each processor? If more than one task is created and given the absence of virtual memory management a direct management of each processor's memory, able to ensure a correct behavior to all tasks allocated on it, is needed. Secondly, the larger is the number of tasks created, the larger is the time required at the beginning for their creation. Finally, it is expected that context switching is an overhead for overall computation. Experiments on small istances of spatial data (which therefore does not require such a memory management) prove this (see figures 4 and 5) and also indicates that, given that one task per processor is better than multiple tasks, the overall optimum number of tasks (i.e. of processors) is four.

This is not surprising since this is the maximum number which allows to the task managing the whole parallel program to directly exchange messages with all other tasks without communication delays.

[1] The communication software is called CSN (Computing Surface Network).

From now on we speak indifferently of task or processor, when it is immaterial the distinction between the software aspect (i.e. the task) and the hardware one (i.e. the processor).

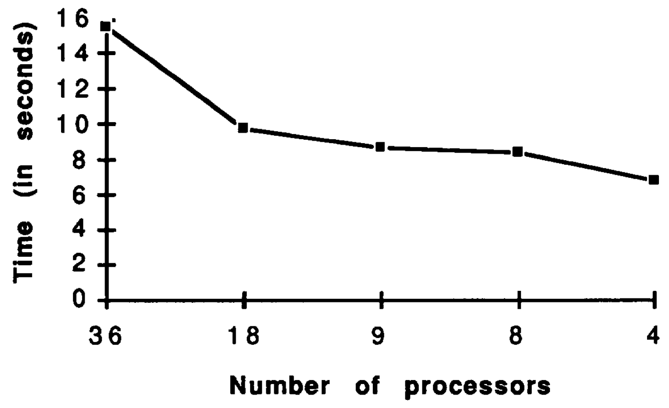

Figure 4: Allocation time vs number of processors.

3.4 Setting for experiments

Time has been measured using machine's clock, which return number of *ticks*, where each tick is 64×10^{-9} seconds (64 nanoseconds). Experiments have been done with the machine completely available, therefore measured time does not include, beyond an unavoidable very small percentage, time spent by the operating system in services unrelated to our computations.

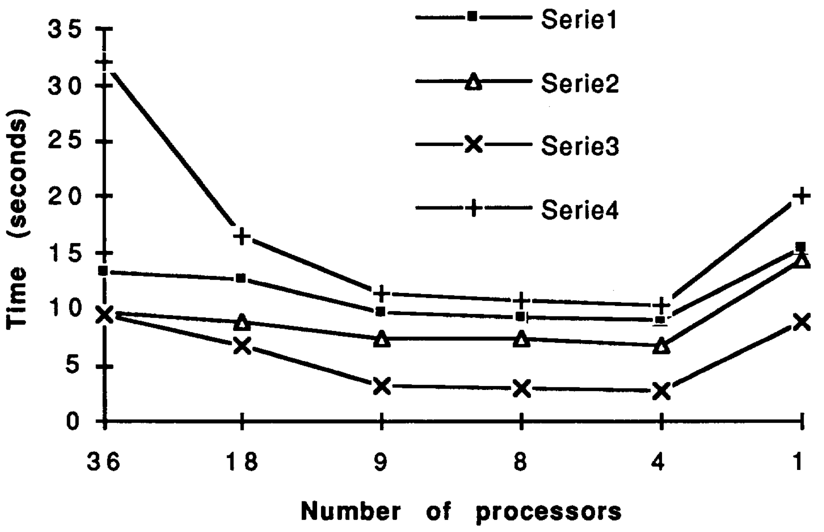

Figure 5: Overall computation time vs number of processors for four different series

4. Architectural considerations

4.1 Adopted software architecture

Given the limitation above described in terms of dynamic creation of tasks the solution adopted in architectural terms is made up by:

- a *scheduler* for sending processing requests to tasks,
- a *server* for receving their answers,
- a *queue* of processing requests, and
- a certain number of *tasks* to carry out the required computations.

The scheduler has the job of servicing the queue, by passing the processing requests to tasks that are available since have finished their previous job. Requests are sent to task using a round-robin policy based on task availability. All tasks communicate with the scheduler using the same communication channel and an asynchronous blocking protorcol.

The server has the job of receiving the results of data processing or the requests for further processing sent back by tasks when finish their current job. It communicates with tasks using a communication channel different from the scheduler's one. All tasks share the same channel and use an asynchronous blocking protocol.

The queue is where all processing requests are stored, with a first-came first-served policy.

Each task is created at the beginning of the parallel computation of the required operation and lives until the operation is completely terminated. It processes the data received as current job and decides if the result can be computed or four new tasks are required. The decision is taken on the basis of the truth table for the operation. It returns either the result of the operation, which may be a long series of nodes in the case marked 'subtree' in the truth table or the parameters for the creation of four new tasks (corresponding to the sons of current node). After returning to the server whichever result it has computed in the current invocation it falls asleep and it is awaken by the scheduler when a new job is ready for it.

4.2 Tuning the architecture

Given that the above described architecture is practically the only feasible solution to implement the parallel approach in the MEIKO machine, the two most important questions are: how many tasks are required for a lowest overall (i.e. initialization plus real processing) computation time? which is the best communication network (in the sense of contributing to reach the lowest possible overall computation time)? These two questions are strictly intertwined and before answer a more detailed analysis of the communication aspects is required.

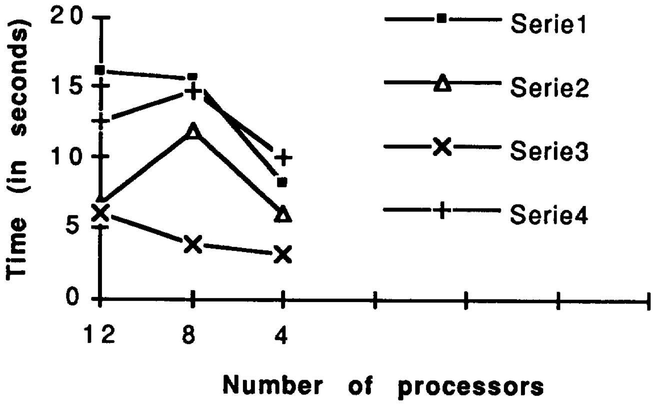

Figure 6: Overall computation time vs number of processors for four different series

From the architectural solutions above it derives that it is easier and more efficient to implement the scheduler and the server within the same task (called simply scheduler in the sequel, for the sake of shortness); this processor therefore requires communication with all the other processors. This fact, plus what we have discussed about the way communication between tasks is implemented, implies that when there are four task communicating directly with the scheduler no delay is introduced by the communication network. Given that computations performed by each task are fairly simple this would suggest the optimum number of processor is four. We did experiments with 12, 8 and 4 tasks, shown in figure 6 (instances of data different from ones used for experiments reported in figure 5 have been used), connected through a communication network with a square grid topology with the scheduler at the center (see figure 7), which show that indeed four is the number of tasks that allows to obtain the lowest overall computation time.

The rationale for this is that 4 is the maximum number of tasks that may be connected to the scheduler at a communication distance equal to 1, that 8 forms a complete square grid with 3 processors on each side, and 12 is maximum number of tasks that may be connected to the scheduler+server at communication distance equal to 2. These experiments show that with more than 4 tasks the overall computation time increases. A more detailed analysis, not reported here, shows that this increase is exactly due to communications. In fact, in this case processors closer to the scheduler have to interrupt their work to forward messages to and from farther processors[1].

4.3 Inizialization

The chosen software architecture requires, as a consequence, that the data structure(s) required for computation are completely sent at the beginning of the parallel computation to each logical processor. The alternative approach, of sending only the portion required for the current step of computation is not feasible, since using the linear quadtree it is not in general possible to know in which position in the list are the sons of the current node. If the scheduler did this search, then the bulk of the tasks' work would be completed and there would be no reason to use parallel processing. Moreover, given the overhead for opening and closing communication channels, it is better to send once-and-for-ever the data each task requires at the very beginning instead of sending them in smaller batches while computation proceeds.

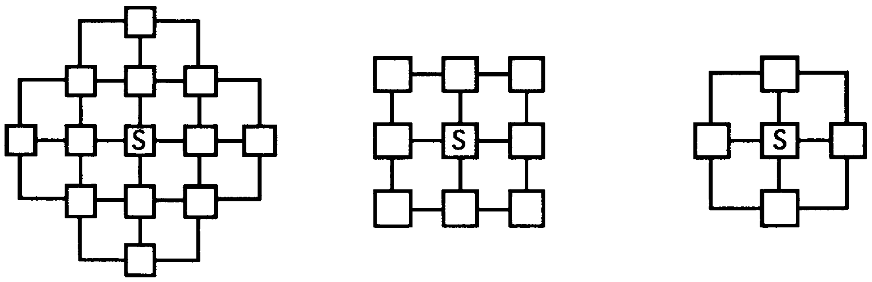

Figure 7: Communication network topologies for 12, 8, and 4 processors.

We have found that the maximum size of messages exchanged between processors is 1024 bytes. With our choice for the data structures, this correspond to 128 elements of the linear quadtree. We have estimated through our experiments that the time required to transmit a buffer with 128 elements is 11620 microseconds, including the overhead for starting and closing the transmission.

In figure 8, below, results of experiments of sending different data structures to processors in the case of 4 processor (3 communication steps), 8 and 12 (4 communication steps) are reported.

4.4 Load balancing

Load balancing is a critical issue in every parallel machine. Given that four is the best number of tasks in our case; the balancing issue is not a very difficult problem, since it can be easily proved that in the average the architectural solution based on queue of requests and round-robin scheduler ensures an homogeneous distribution of load to each processor. Experiments prove that this is true also in practice. Load is defined in terms of time spent by each task working on data, not simply in terms of number of activations of tasks, since work done during each activation may vary.

----

(1) We have been able to determine that when a communication channel is required at the same time by more two tasks to communicate with the scheduler, precedence is given to the farther task, and the closer one has to wait.

4.5 Speed-up and efficiency

We recall that the *speed-up* of a parallel algorithms is measured as the ratio between the time required to the best sequential algorithm running on a single processing unit of the parallel machine to complete the job and the time required to the parallel algorithm to complete the same job using a certain number of processors. The *efficiency* is the ratio between the obtained speed-up and the number of processors used.

We have measured speed-up and efficiency for a number of different data sets (see figure 9). Obviously, only the case of four tasks has been examined. Results are shown in figure below. Note that since both union and intersection have the same algorithmic structure, reported result are valid for both operations.

5. The performance evaluation model

5.1 Initialization issues

As discussed above, data structures are completely sent to each task at the beginning of the whole process. We derive here a model for estimating the initialization time. It is made up by two components, that is T_{all}, i.e. the time required to the environment to create scheduler, server, and task, to load them into memory and to start them, and T_{rst}, i.e. the time required to send data structures to all tasks.

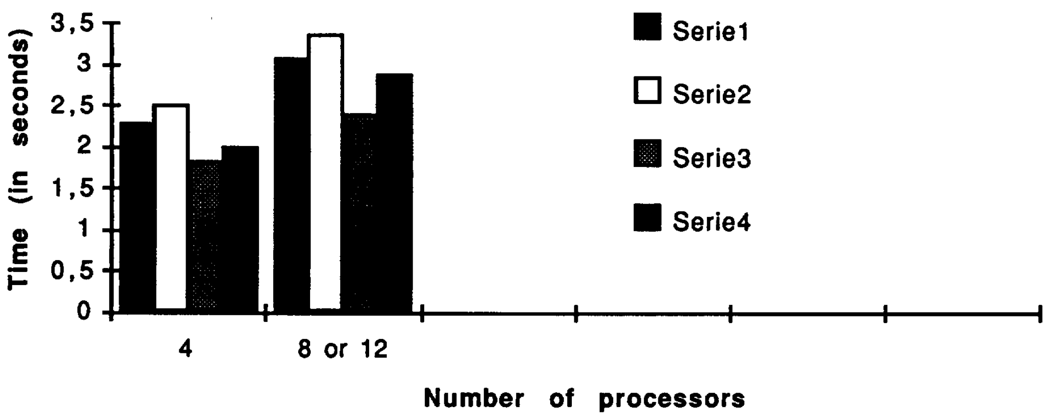

Figure 8: Initialization time vs number of processors.

Let a *C_4-network* denote a communication network where each processor is connected to exactly four other processors. Each processor can send only one data item to exactly one other processor during a step. Assume a processor in such a network wants to communicate one data item to all the other processors and this is the only communication activity in the network.

# leaves in the result	Time (ms) sequential	Time (ms) parallel	Speed-up	4 procs Efficiency
4048	4,154	3,04	1,37	34,16%
5539	14,3	5,7	2,51	62,72%
7666	13,9	6,7	2,07	51,87%
8095	20,18	9,9	2,04	50,96%
10171	15,46	8,13	1,90	47,54%

Figure 9: Speed-up and efficiency

Lemma 1. The number of processors P_k which after k steps have received one data item originated at a single point in a C_4-network satisfies the following recurrence relation:

$$P_k = \begin{cases} 2^k & 0 \leq k \leq 4 \\ 2P_{k-1} - P_{k-4} & k > 4 \end{cases}$$

Proof (draft): Any processor, but the one originating the message, has used all its communication links after having sent three messages. Therefore, at each step, the number of processors that has received the data item is the number of processors that had received it at previous step plus the number of those that has received it in this step. But the latter number is equal to the number of those that had received it at previous step minus the number of those that received it four steps before (since those have now used all the channels).

CVD

The following theorems characterize the above recurrence.

Theorem 1: $P_k = O(2^k)$.

Proof (draft): We can approximate P_k by above as $P_k \leq 2P_{k-1}$.

CVD

Theorem 2: $P_k = \Omega((9/5)^k)$.

Proof (draft): We can approximate P_k by below as $q^k \leq P_k$ and then calculate that q=9/5.

CVD

Let L_1 and L_2 denote the number of leaves of the two linear quadtrees to be processed. Then $F=\left(\left\lceil \frac{L_1}{128} \right\rceil + \left\lceil \frac{L_2}{128} \right\rceil\right)$ buffers have to be sent to all processors. The time, expresses in microseconds, required to initialize Q processors (T_{rst}) is given by the following corollary:

Corollary 1: $11620\ [4(F-1) + \lg_2 Q] \leq T_{rst} \leq 11620\ [4(F-1) + \lg_{9/5} Q]$.

CVD

5.2 Model parameters

The model for evaluating processing time is described in terms of a number of parameters. Some of them describe the behavior of the scheduler, others refer to tasks. Let us denote with N the number of tasks. From the discussion above about the best software architecture, it follows that we have studied the case N=4, but the result maintain their validity for all N≤4.

Parameters for the scheduler are the following:

T_{stx}:	it is the total time required by the scheduler to send all data items to the tasks;
T_{srx}:	it is the total time required by the server to receive all data items from the tasks;
T_{sat}:	it is the total time the server waits for result from the tasks;
T_{sut}:	it is the total time the scheduler performs useful computations.

Parameters for describing a task are now introduced. They are averaged across tasks.

T_{trs}:	it is the total time a task wait to start its next job; it includes the real waiting time (called sleep in the following and due to the fact that scheduler is busy with other tasks) and the total time used to receive data items from the scheduler, i.e. $T_{trs}=T_{tso}+T_{trx}$ (see below);
T_{tso}:	it is the total time a task sleeps before receiving data items describing its next job;
T_{trx}	it is the total time a task spends to receive data items describing its next job; it is equal to the total time required by the scheduler to send all data items to the tasks divided by the number of tasks, i.e. $T_{trx} = T_{stx}/N$;
T_{tut}:	it is the total time a task performs computation useful for the operation;
T_{ttx}:	it is the total time required by a task to send all its results back to the scheduler.

The total time required for processing is the sum of the four times introduced above for the scheduler, and is equal to the sum of the analogous times for each task:

$$T_{stx} + T_{srx} + T_{sat} + T_{sut} = T_{tso} + T_{trx} + T_{tut} + T_{ttx}$$

5.3 Identification

In this section we describe in which way we have been able to estimate a value for each of the parameters. The analysis we have done are based on the analysis of:

- the structure of the algorithms for union and intersection,
- the way scheduler and server works, and
- the policy for servicing the queue.

Hypothesis made on the basis of this analysis have been then verified by experiments where value for constants have been derived. Time is expressed in micro-seconds (μsec).

In the analysis we speak of:

- *grey-grey couples* to indicate the couples of nodes where both nodes are grey, and GG denotes the number of such couples;
- *colour-colour couples* to indicate the couples of nodes where both nodes are different from grey (that is nodes are either white or black), and CC denotes the number of such couples;
- *grey-colour couples* to indicate the couples of nodes where one node is grey and the other is different from grey (white or black is the same), and GC denotes the number of such couples, and DGC denotes the number of descendants of the grey node in such a couple.

• T_{stx}

This time is directly related to the number of grey-grey couples. It is only in this case, infact, that recursion is needed and four new tasks have to be created. The scheduler therefore executes four transmissions to tasks for each grey-grey couple.

Let us denote with GG the number of grey-grey couples. We have derived from the experiments the following relation:

$$T_{stx} = 176.4\ (\ GG * 4\ )\ .$$

• T_{srx}

This time is directly related to three factors:

- the number of grey-grey couples; for each of these couples, in fact, four data items are received (corresponding to parameters for the activation of four new tasks);
- the number of colour-colour couples; for each of these couples, in fact, one data item is received corresponding to the result of the operation (union or intersection);
- the number of grey-colour couples; once the first of the descendants of a grey node is found (which requires time proportional to $\frac{1}{2}\frac{L_1+L_2}{2}$ since we use linear search), for each couple a number of data items are received corresponding to all the nodes in the subtree rooted at the grey node and must be added to the list of results. Moreover, we need to consider that in the case of union the subtree is returned when white-grey couples are found, while for the intersection this role is

played by the black-grey ones. We can estimate that in the average white-grey couples and black-grey couples equally divide the number of grey-grey couples[1].

From the discussion above and the experiments the following relation derives:

$$T_{srx} = 101.5\ (\ GG * 4) + 102.2 * CC + \left(0.88\ \frac{1}{2}\frac{L_1+L_2}{2} + 227 * DGC\right)\frac{GC}{2}\ .$$

Note that the two constants 101.5 and 102.2 are very close as it should be, since they refers to transmission time of simple data items of the same kind (theoretically they should be equal).

- T_{sat}

To derive a formula for waiting time of the scheduler we need to consider that it may depend only by T_{tut} and by T_{tso}, since during task transmission and reception phases the scheduler is busy in communication. A precise derivation of such a dependance is highly complex since requires to model the distribution of waiting, transmission and processing times and their interactions for all the tasks. From experiments we have derived a simpler formula, reported below:

$$T_{sat} = 0.56\ (\ T_{tut} + T_{tso}\ )\ .$$

- T_{sut}

This time is directly related to the three factors below described.

- The number of grey-grey couples: each of these couples generates four data items in the queue; they have to be deleted from the queue and prepared for the delivery to the tasks.
- The number of colour-colour couples: for each of these couples, one data item in the queue containing the result sent by a task must be delcted from the queue and stored in the list of results.
- The number of grey-colour couples: for each of these couples, a variable number of data items is received by the scheduler and added to the list of results; time is required to prepare such a reception. Like in the discussion for the analogous factor in the case of T_{srx} also in this case, depending on the type of operation, it is the number of white-grey couples or black-grey ones which counts.

From the discussion above and the experiments the following relation derives:

$$T_{sut} = 261.4\ (\ GG * 4) + 327.4 * CC + 280.8 * DGC\ \frac{GC}{2}\ .$$

- T_{tso}

The time each task spend sleeping is usually short, since both computation time and communication time in the scheduler are reasonably short. But when a task is sending a long list of nodes back to the scheduler as result, tasks that have already finished their work and are waiting for a new jobs sleeps for all the time required by the scheduler to receive such a transmission. We can reasonably assume, to simplify things, that sleeping deriving from short transmissions is negligible and that sleeping times are distributed so that no more than one task at a time is sleeping.

From a detailed analysis of data we have derived that 44 elements is the boundary condition between long deliveries and short ones. Let us denote with GC' the number of times such a long delivery happens, that is anytime a subtree of the grey node in a grey-colour couple contains more than 44 nodes, and with DGC' the number of the descendants of the grey node in such a case. Then we can estimate this time using a relation with the same structure as the one used for the last term of T_{srx} :

$$T_{tso} = \left(0.88\ \frac{1}{2}\frac{L_1+L_2}{2} + 227 * DGC'\right)\frac{GC'}{2}\ .$$

- T_{tut}

This time is directly related to the two factors described below.

- The number of grey-grey couples: this affects T_{tut} in two ways. Firstly, because in each of the four newly activated tasks, the path received must be find in the lists of leaves of both quadtrees. Secondly, because when a grey-grey couple is found it has to be managed by preparing data to be sent back to the scheduler.
- The number of grey-colour couples: in this case either the couple enables the task to immediately compute the result for the considered operation (i.e. it is a black-grey couple for union , or a white-grey for intersection) or requires the task to prepare the transmission of a list of leaves as result.

From this discussion and the experiments, the following relations derive:

$$T_{tut} = (2 * 0.88)\left(\frac{1}{2}\frac{L_1+L_2}{2}\right)(\ GG * 4\ ) + 12.9 * GG + $$
$$+ \left(0.88\ \frac{1}{2}\frac{L_1+L_2}{2} + (227\text{-}101.85) * DGC\right)\frac{GC}{2}\ .$$

The factor (227-101.85) considers that here only the time for moving on the list has the to be considered (101.85 is the average time for transmitting one data item).

- T_{ttx}

Note that it is less than the total time required by the server to receive all data items from the tasks since, from

[1] Note that since grey nodes are not in the linear quadtree we discover that a node is grey node by arriving at the end of the list without having found it. Therefore, to find the first of the descendants list has to be searched a second time: in the meantime scheduler has been told that a grey-colour couple has been found and is waiting.

the discussion above about T_{srx}, it is clear that scheduler waits while the task find the first data item in a subtree of a grey node, and waits white the task is moving along the list of leaves.

$$T_{ttx} = T_{srx} - \left(0.88 \frac{1}{2}\frac{L_1+L_2}{2} + (227-101.85) * DGC\right)\frac{GC}{2}.$$

6. Conclusions

In this paper we have defined and tested through experiments a model for evaluating performances of a message passing architecture parallel machine in the context of spatial data processing. 2-dimensional data of the type "region" have been considered under the operations of union and intersection.

The model is able to predict the overall processing time needed to execute union or intersection operation between two regions of arbitrary shape as a function of a small number of parameters describing input data. The model obtained is of general validity, and is the first necessary step in tackling the issue of query optimization for spatial data in a parallel environment.

Refinements to this work will be addressed to optimize algorithms by eliminating all useless waiting times and improving operations on list of leaves (e.g. binary instead linear search ...), to refine the model for what regards the scheduler's waiting time, to include collisions on the communication channels in the model, which is useful for a number of tasks N>4, even if this is not a really interesting case since the overall processing time increases.

More interesting is to examine the possibility of estimating "a priori" some of the model input parameters on the basis of a probabilistic model of the data. Extensions will also be made towards a more general model to be used for query optimization purposes in systems for spatial data management, which is one of the most important research topic in this field [Gün90].

Acknowledgments

We thank Ettore Apolloni, Franco Arcieri, Alberto Postiglione, and Maurizio Talamo for useful discussions, in the context of ITU-LAND project, related to parallel spatial data processing. Thanks also to the MEIKO machine's technical support people at Advanced Computing System in Milan for their useful information on the experimentation environment.

References

[Abe83] D.J. Abel: "A B+-tree Structure for Large Quadtrees", Computer Vision, Graphics and Image Processing 27, 1 (July 1984), pp. 19-31.

[Arc92] ARCIERI F., NARDELLI E.: "An integration approach to the management of geographical information: Cartech", 2nd ACM-IEEE International Conference on Systems Integration (ICSI'92), Morristown, NewJersey, USA, Giugno 1992.

[Arc93b] ARCIERI F., GIORDANI S., NARDELLI E.: "An interaction based approach to the use of geographical information systems for regional planning", International Conference on Systems Engineering (ICSE'93), Las Vegas, Nevada, USA, Luglio 1993.

[Bes92] T.Bestul: "Parallel paradygms and practices for spatial data", Center for Automation Research, Univ. of Maryland, CAR-TR-624, CS-TR-2897, April 1992

[Gar82] I.Gargantini: "An Effective Way to Represent Quadtrees", Comm. of the ACM, 25, 12, 1982, pp. 905-910.

[Gar91a] GARGANO M., NARDELLI E., TALAMO M.: "Abstract Data Types for the logical modeling of complex data", Information Systems, 16, 6, 1991.

[Gar91b] GARGANO M., NARDELLI E., TALAMO M.: "A model for complex data: completeness and soundness properties", International Workshop on Database Management Systems for Geographical Applications, Capri, Maggio 1991, in G.Gambosi, M.Scholl, H.W.Six, (eds), Geographic Database Management Systems, Esprit Basic Research Series, Springer Verlag, Giugno 1992.

[Gün90] O.Günther, A.Buchmann, Research issues in spatial databases, SIGMOD Record, 19, 4, 1990.

[Güt92] R.H.Güting, M.Schneider, Realms: a foundation for spatial data types in database systems, Fernuniversität Hagen, Report 134, 11/1992, also in Proc. 3rd Intl. Symp. on the Design and Implementation of Large Spatial Databases, Singapore, 1993.

[ITU92] ITU-LAND Project, Special Action in the Esprit Basic Research Programme, Technical Annex, November 1992.

[Chi90] C.H.Chien, T.Kanade: "Distributed quadtree processing", 1st Intl. Symp. on the Design and Implementation of Large Spatial Databases, Santa Barbara, Ca., 1990.

[Nar93a] E.Nardelli, G.Proietti: "Efficient Secondary Memory Processing of Window Queries on Spatial Data", Int. Symposium on Computer and Information Sciences, Istanbul, Turkey, November 1993.

[Nar93b] E.Nardelli, G.Proietti: "A unifying probabilistic model for spatial data represented by quadtrees", Tech. Rep. 38, Dept. of Pure and Applied Mathematics, Univ. of L'Aquila, November 1993.

[Nie84] J.Nievergelt, H.Hinterberger, K.C.Sevcik: "The grid file: an adaptable, symmetric multikey", Tech. Rep. 38, Dept. of Pure and Applied Mathematics, Univ. of L'Aquila, November 1993.

[Kam92] I.Kamel, C.Faloutsos, "Parallel R-trees", Proc. of 1992 Conference on Management of Data (SIGMOD92), S.Francisco, May 92.

[Sam90] H.Samet, The design and analysis of spatial data structures, Addison Wesley, 1990.