Parallelization of the N-queens problem. Load unbalance analysis.

Laura De Giusti[1], Pablo Novarini[2], Marcelo Naiouf [3] ,Armando De Giusti[4]

Research Institute on Computer Science LIDI (III-LIDI) [5]
Faculty of Computer Science - National University of La Plata

Abstract
The paper presents an analysis of three parallelization structures of the *N*-queens problem, taking into account *N* processors. The focus has been set on investigating the adaptation of the architecture structure to the proposed algorithm type, so as to study the load unbalance in each case, for which two different metrics have been established.
The experimental results and the efficient implementation of the algorithms are discussed together with the related current research lines.

Key words: Parallel Systems. Parallel algorithms. Load Balance. Complexity.

Introduction
The increasing importance and interest in parallel processing within Computer Science is clear for several reasons. Generally speaking, parallel machines allow to solve problems of increasing complexity and obtain results faster, and in several cases, they represent the only viable choice since sequential solutions involve unacceptable times. Besides providing faster solutions, parallel applications are capable of solving larger, more complex problems whose input data or intermediate results exceed a CPU memory capacity; simulations can be run at finer resolution, and the physical phenomena can be modeled more realistically [1] [2].

It is important to refer to a parallel algorithm not in an isolated manner but together with the computing model for which it was designed. Unlike sequential computation, where Random Access Machine (RAM) is practically accepted as a standard, in parallelism there does not exist a unifying theoretical model (since each emphasizes certain aspects over others) and there exists a broad diversity of platforms. On the other hand, for each application, there could be an optimal machine, various implementation alternatives with homogeneous or heterogeneous hardware, with tight and loose coupling of its components. Thereby, *parallel systems* are referred to as the combination of algorithm and architecture [3] [4].

[1] *UNLP Scholar. Part-time Teaching Assistant. Faculty of Computer Science. UNLP. ldgiusti@lidi.info.unlp.edu.ar*
[2] *Part-time Teaching Assistant. Faculty of Computer Science. UNLP. pnovarini@info.unlp.edu.ar*
[3] *Chair Professor. Faculty of Computer Science. UNLP. mnaiouf@lidi.info.unlp.edu.ar*
[4] *Principal Researcher of CONICET. Full-time Chair Professor. Faculty of Computer Science. UNLP. degiusti@info.unlp.edu.ar*
[5] *III-LIDI member of Research Institute on Computer Science and Technology (IICyTI)TE/Fax +(54)(221)422-7707. http://lidi.info.unlp.edu.ar*

A parallel application defines a set of intercommunicated components that should be assigned in the physical resources of the target architecture. The last step in the development of parallel algorithms is the process mapping over processors; the objectives aim at optimizing the use of processors and obtaining the best response time of the application, carrying out the work distribution to the processors so that the computational load tends to be equal (*balanced*) in time [5] [6] [7].

This is one of the core aspects of parallel processing, since it has a direct impact on the efficient use of resources (that imply costs) and on the achievable performance improvement.

Within the types of problem, a distinction can be made between those whose nature allows a parallelization so that the obtained load balance is near optimal (generally, those with a regular execution pattern, such as some solutions to the matrix multiplication problem), and those where the execution pattern is irregular or has a dynamic nature or is data-dependant (where the load balance objective is harder to achieve) [8] [9] [10].

This paper aims at analyzing the load balance obtainable in the parallelization of a not basically balanced problem. The *N*-queens problem belongs to this type, and it has thus been chosen for the analysis.

N-queens Problem. Sequential Solution.

The *N*-queens problem is a generalization of the well-known 8-queens problem, which consists in arranging 8 queens on a chessboard so that none can take another. A queen attacks another if they are located on the same diagonal, row or column.

In the case of the *N*-queens, *N* queens are placed on a *N*x*N* board. There exists a known number of solutions; for instance, there are 92 solutions for placing 8 queens on a 8x8 board [11] [12].

Other problems are derivable from this. Among them, it is worth mentioning the problem which, given a *N*x*N* board, looks for the lesser quantity of queens that can be placed so that all the board squares are attacked by some queen [13].

An initial solution to the *N*-queens problem, by way of a sequential algorithm, consists in testing all the possible placement combinations of queens on the board and choosing the valid ones.

The combination in which no queen of the board is attacked by another is considered as valid. This solution can be upgraded by discarding, during the search, those ways by which a solution to the problem cannot be found [14].

The pseudo-code of the sequential solution is as follows:

```
beginArray ()  //diagonals and columns marking them empty
call to addQueen proceeding

addQueen()  //place a queen on the following row
  row++
  for each column do(i:1..N)
        test if a queen can be placed on column i.
        If true then
             mark the column and diagonals as filled.
             If is the last row then
                 New solution found
             If not
                  Call addQueen proceeding

        Unmark the column and diagonals for testing other combinations.
```

The solution presented above shows a non-linear growth in the complexity as the size of the board increases.

Load Balance for the *N*-queens problem

This paper aims at analyzing the load balance in the parallelization of the *N*-queens problem.
The metrics used for the *load* is function of the total quantity of analyzed squares to place some queen (l_i).

In order to measure the load unbalance, two metrics were used:

1.- Unbalance between maximum and minimum load:
This metrics considers the relation between the maximum work load and the minimum load of processors.

$$M_1 = C_{\max} / C_{\min} \qquad (1)$$

For instance, if C_{max} = 100 and C_{min} = 50, M_1 = 2.
This means that the processor that works more does twice as much work as that which works less.

2.- Unbalance related to the average work done:
This metrics takes into account the deviation percentage of the work done by the processors in relation to the average of the work done.

$$M_2 = (P*100) / T_{prom} \quad where \;\; P = \sum_{i=1}^{N} \left| T_i - T_{prom} \right| / N \qquad (2)$$

For instance:
if M_2 = 0 then the obtained balance is the optimal.
if M_2 = 50 it means that each processor deviates a 50 % of the work that it should carry out if it had an optimal balance.

Parallelization of the *N*-queens problem

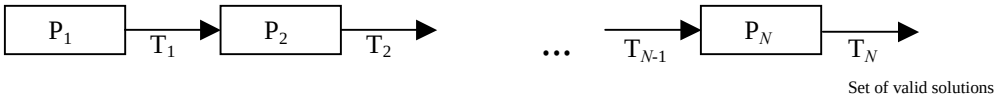
All the algorithms presented below perform *N* stages where, in each stage *I,* they try to place queens on all the valid position of row *i*.

1- Solution with processor pipe:

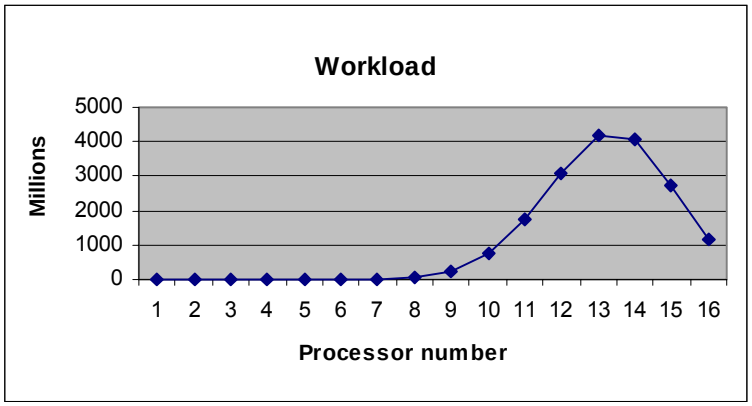
Let $P_1..P_N$ be processors and T_i the set of boards with queens placed on valid positions on the first *i* rows. A processor P_i is in charge of placing the queens on row *i*.

The work begins with processor P_1, which places a queen on each possible position of row 1, thus obtaining a set of initial boards (T_1). Each of these obtained boards passes to processor P_2, which places a queen on all the valid positions of row 2 in each of them, passing the obtained boards (T_2) to the next processor.
The process is repeated until *N* processor is reached. This receives from processor P_{N-1} the boards with queens located accurately on the first *N*-1 rows (T_{N-1}), and its task is to place the queens on row *N*, in order to obtain the set of solutions to the problem (T_N).

A graphical representation of the architecture is as follows:



The following graphic shows the quantity of work done by each processor for a 16 (*N*=16) sized board. To the right, there is a chart with the work load of each processor.



Processor nª	Work Load (l_i)
1	16
2	256
3	3,360
4	35,776
5	315,008
6	2,268,992
7	13,421,056
8	63,975,296
9	245,195,328
10	741,742,016
11	1,735,663,456
12	3,102,285,760
13	4,164,854,528
14	4,062,362,112
15	2,738,528,288
16	1,152,033,408

Next, the two metrics defined in (1) and (2) are analyzed for different number of processors:

N	M_1	M_2
8	568	80.55
10	9,632	89.38
12	222,720	99.38
14	6,471,872	105.82
16	260,303,408	113.15

It is clear that the load unbalance is unacceptable for increasing *N*, and this is typical of the characteristic of the problem (breadth first search).

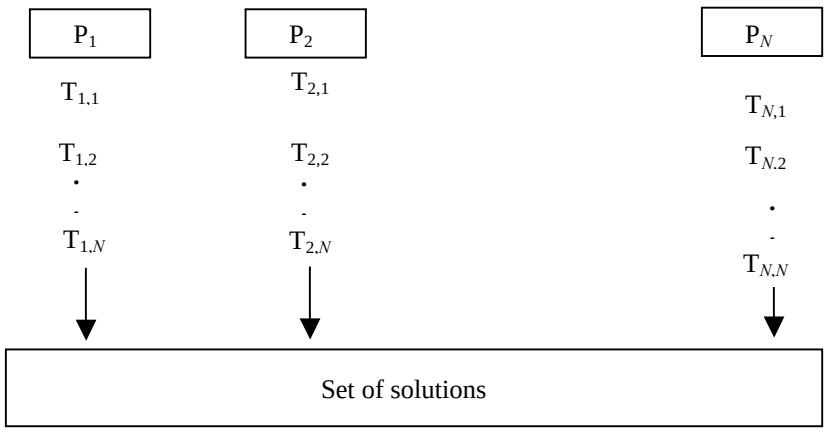
2.-Loosely coupled processor N Solution:

Let $P_1..P_N$ be processors and $T_{i,j}$ the set of boards where *i* represents the column in which the queen was placed on the first row, and *j* the row up to which the board has queens.

For instance: $T_{2,4}$ is the set of all the boards with queens placed accurately on the first four rows, being the queen of the first row placed on the second column.

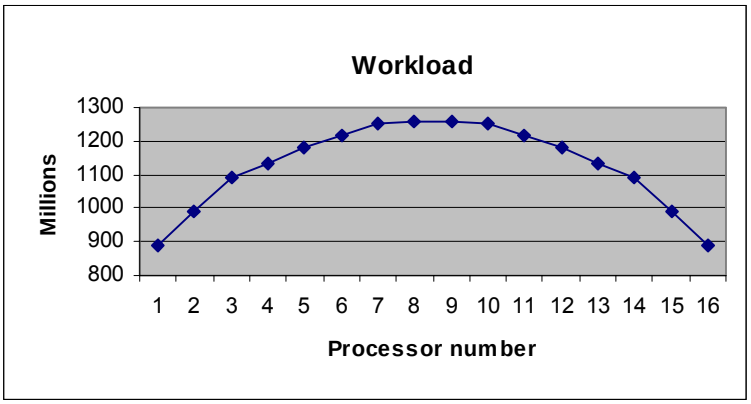
A processor P_i is in charge of finding the set of boards ($T_{i,N}$) with all the possible solutions having placed the queen on the first row column *i*. In this case there are no communication between the processors during the search of the solutions. Each processor works independently.

A graphical representation of the architecture is the following:



An equivalence with the previous notation is given by: $\bigcup_{j=1}^{N} T_{j,1} = T_1$

The following graphic shows the quantity of work done by each processor for a 16 (*N*=16) sized board. To the right, there is a chart with the work load of each processor.



Processor n[a]	Work Load (l_i)
1	887,143,505
2	992,762,433
3	1,089,991,889
4	1,133,623,105
5	1,182,782,369
6	1,216,458,385
7	1,249,758,561
8	1,258,822,081
9	1,258,822,081
10	1,249,758,561
11	1,216,458,385
12	1,182,782,369
13	1,133,623,105
14	1,089,991,889
15	992,762,433
16	887,143,505

If the analysis of the two defined unbalance metrics is now repeated,

N	M_1	M_2
8	1,15	4,58
10	1,16	4,03
12	1,22	5,41
14	1,31	7,01
16	1,41	9,08

a remarkable decrease in the load unbalance can be noticed. For instance, for 16 processors the “average” unbalance - though with metrics M1 it reaches the 41% between the maximum load processor and that of minimal load - is only of 9%.

3.- Parallel pipe variant:

Noting the work unbalance carried out in the first solution (using a processors pipe), it can be said that the work to be done by each processor varies according to the corresponding row in which it places the queens.

The processors that place queens on the central rows of the board carry out a greater work load, since they count with more possible combinations to test.

The algorithm represented below attempts to balance the work done by each of the processors used in the solution of the problem.

Let $P_1..P_N$ be processors and $T_{i,j}$ the set of boards.
Each processor P_i begins a solution placing a queen on column *i* of row 1, $T_{i,j}$,

In each step of the algorithm, processor P_i receives from processor P_{i-1} a set of boards $T_{k,j}$, and places a queen on each valid position of the next row, thus obtaining the set of boards $T_{k,j+1}$, which are in turn passed to processor P_{i+1}.

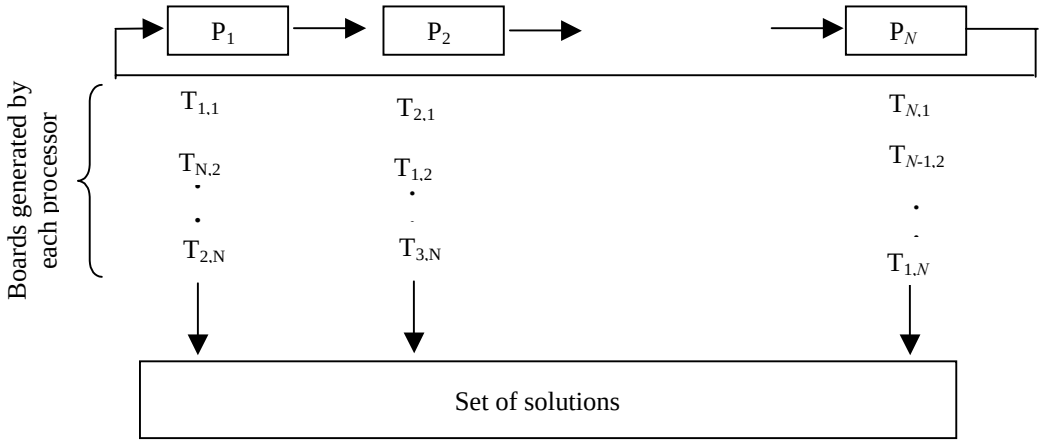
For instance, processor P_1 starts its work placing a queen on row 1, column 1, and send its board ($T_{1,1}$) to processor P_2. It then places a queen on each valid position of row 2 for the solution started by P_N, obtaining $T_{N,2}$. Next, it receives from processor P_N, boards with two queens placed on the first two rows (generated by P_{N-1}), and it places queens on each of them on the valid positions of row 3, (obtaining $T_{N,3}$), and so on.

Thus, each processor carries out in each stage *j* the work equivalent to that of processor *j* of the pipe solution. Thereby each processor plays all the roles of the first solution processors.

With this, an improvement on the load balance is achieved by way of balancing each processors' work. All the processors place queens on all the rows.

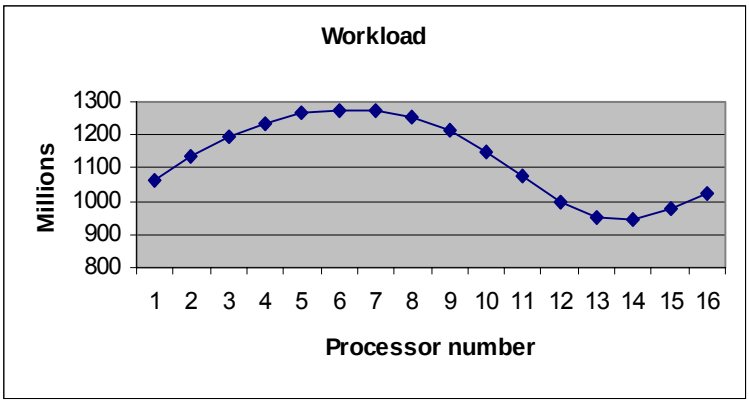
Note that ***without*** *changing the physical and logical architecture* of the solution with the processors pipe, and ***without*** *essentially changing the algorithm*, an important change is achieved in the load balance.

A graphical representation:



Each processor P_i has the set of boards $T_{i+1,N}$ that represents the solution initiated by processor P_{i+1}, which placed the queen of the first row on column *i*+1.

The following graphic shows the quantity of work done by each processor for a 16 (*N*=16) sized board. To the right, there is a chart with the work load of each processor.



Processor nª	Work Load (l_i)
1	1,061,848,177
2	1,134,981,281
3	1,194,115,313
4	1,235,277,169
5	1,264,375,793
6	1,274,235,233
7	1,273,619,297
8	1,251,118,593
9	1,216,519,633
10	1,150,072,401
11	1,076,401,825
12	999,309,057
13	949,764,977
14	943,807,137
15	974,697,313
16	1,022,541,457

N	M_1	M_2
8	1.23	8.14
10	1.23	6.61
12	1.24	6.94
14	1.28	8.21
16	1.35	9.50

The unbalance metrics M_1, compared to the previous parallel solution, tends to be better in this solution for increasing N, while metrics M_2 shows similar results (by definition, it averages the sum of unbalances).

Conclusions and future work lines

The study of the load unbalance has been initiated for a type of parallel systems, focusing on the adjustment of the algorithm to the supporting architecture.
An important result is the development of a new parallel algorithm over a processor pipe for the *N*-queens problem, which tends to balance the load for increasing *N*.

There exist several open research lines with this respect:

- To analyze in depth the measurements for increasing N.
- To study the communications incidence in each case.
- To experiment on distributed shared memory architectures.
- To experiment on multiprocessor structures with *M* processors (*M* < *N*).
- To analyze the effect (and the compensation potential) of the processors heterogeneity.

Bibliography

[1]Andrews G., “Concurrent Programming: Principles and Practice”, The Benjamin/Cummings Publishing, Inc, Andrews G., “Foundations on Multithread and Distributed Programming”, Addison Wesley, 1999.

[2]Coffin M., “Parallel programming- A new approach”, Prentice Hall, Englewood Cliffs, 1992.

[3]Hwang K., Xu Z., “Scalable Parallel Computing”, McGraw-Hill, 1998.

[4]Keller J., Kebler C., Traff J., “Practical PRAM Programming”. A Wiley – Interscience publication. 2001.

[5]Quinn M. “Parallel Computing Theory and Practice”. McGraw – Hill. 1994.

[6]Watts J., Taylor S., “A Practical Approach to Dynamic Load Balancing”, IEEE Transactions on Parallel and Distributed Systems, 9(3), March 1998, pp. 235-248.

[7] Kumar V., Grama A., Gupta A., Karypis G., “Introduction to Parallel Computing. Design and Analysis of Algorithms”, The Benjamin/Cummings Pub. Company, Inc., 1994.

[8]Bruen A., Dixon R., “The n-queens Problem. Discrete Mathematics”. 12:393-395, 1997.

[9]Hedetniemi S., Hedetniemi T., Reynolds R. “Combinatorial problems on chessboards: II”. Chapter 6 in Domination in graphs: advanced topic, pp. 133-162, 1998.

[10]Bernhardsson B., ”Explicit Solution to the n-queens Problems for all n”. ACM SIGART Bulletin,2:7,1991.

[11]http://www.wi.leidenuniv.nl/~kosters/nqueens.html

[12]http://www.rain.org/~mkummel/stumpers/8queens.txt

[13]http://www.jsomers.com/nqueen_demo/nqueens.html

[14]http://www.funducode.com/freec/recursion/recursion3.htm