EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN - PS DIVISION

CERN/PS 96-47 (PA)

DEVELOPMENT OF PARALLEL CODES FOR THE STUDY OF NON-LINEAR BEAM DYNAMICS

M. Giovannozzi[1,c]
E. McIntosh[2]

Abstract

We consider the nonlinear transverse motion in a circular particle accelerator. A key parameter for accelerator performance is the so-called dynamic aperture. This quantity is defined as the volume in phase space of the stable initial conditions. The accurate numerical computation of such a volume is very CPU time-consuming.
In this paper we present some original parallel algorithms to speed-up the evaluation of the dynamic aperture. A detailed analysis of different algorithms implemented is carried out.
Furthermore, we studied the dependence of the CPU-time on the phase space parameters as well as the load balancing of the proposed techniques.

1) INFN, Sezione di Bologna, V. Irnerio 46, Bologna, I-40126, Italy
2) CERN-CN Division
c) Giovannozzi@bologna.infn.it

Submitted to International Journal of Modern Physics C

Work partially supported by EC Human Capital and Mobility contract

Geneva, Switzerland
10/12/96

1 INTRODUCTION

The search for confirmation of the Standard Model relies heavily on the new generation of hadron colliders, such as the planned LHC[1]. Higher and higher energy beams are required to produce the particles predicted by the theories of fundamental interactions, and a very intense bending magnetic field is necessary to confine these high-energy beams in a circular machine. The only way to generate strong fields is to use superconducting magnets. Unfortunately, this technology has one main drawback (apart from the complex cryogenics needed): it is not possible to design a superconducting magnet producing a high-quality field. This means, for instance, that a dipole generates a magnetic field which, in addition to the required uniform field, includes nonlinear multipolar components.

The particles circulating in superconducting accelerators experience nonlinear forces which produce strong instabilities and losses. These effects could prevent safe operation of the machine for two main reasons. Firstly, the superconducting magnets are damaged by the energy deposited by any particles hitting the beam pipe and may quench. Secondly, the luminosity of the machine, which is proportional to the intensity of the circulating beam, decreases, thus reducing the rate of production of events.

As a consequence, the overall performance of the machine is strongly reduced. The main parameter to quantify these harmful effects is the so-called dynamic aperture (hereafter DA); this is the volume in phase space in which stable motion occurs. A large dynamic aperture implies a wide stable region where one can operate with the beam without experiencing particle losses.

In a previous paper[2], we analysed the problem of computing the DA in the presence of strong nonlinear perturbations. Some algorithms were described and numerical simulations were carried out to test the precision of the proposed techniques. Particular emphasis was devoted to the problem of the accuracy of the computations due to the discretization of the phase space parameters, but no attention was paid to the possibility of reducing the CPU-time by using non-conventional techniques. All these tools were implemented in PLATO (Perturbative Lattice Analysis and Tracking tOols)[3], a program library developed to analyse nonlinear phenomena in accelerator physics.

To further improve the efficiency in the computation of the dynamic aperture, in the sense of improving the accuracy and/or the CPU-time needed by the calculations, a promising approach is to convert the sequential tools developed so far into parallel algorithms.

Nowadays, computer systems with a relatively large number of processors (10 to 100) are widely available, either as workstation clusters, shared memory multi-processor machines, or scalable parallel-processing systems with physically distributed memory. At CERN, for instance, there are several such systems of each type and, in particular, there is a MEIKO CS-2, a scalable parallel computer with 128 processors. This machine has been developed with the support of the European Union, in the framework of the ESPRIT III Programme, and installed at CERN since 1994.

Parallel computation could open up new perspectives in the field of accelerator physics. In particular this approach could be used to solve the problem of sorting efficiently the magnets for LHC[4, 5] and to simulate accurately the beam dynamics under the influence of space charge effects[6, 7, 8], just to mention two subjects of research.

Both problems are computationally intensive. The first case is an optimisation problem: many different configurations of the machine have to be analysed in order to find the best solution. The second involves the calculation of the beam evolution under the effect of the electro-magnetic interactions between particles inside the beam and between particles

and the environment, in addition to the usual magnetic forces due to the beam interaction with the lattice of the machine.
The paper is organised as follows. In Section 2 we review the theory developed to compute the dynamic aperture, presenting two different approaches. In Section 3 we discuss the parallel implementation of the previous algorithms, while in Section 4 we give an overview of the parallel machine used for the study presented in this paper. In Section 5 we present the results concerning the performance of the different algorithms with respect to the CPU-time, the load balancing and other issues. Finally, some conclusions are drawn in Section 6.

2 DYNAMIC APERTURE: DEFINITION AND NUMERICAL COMPUTATION

The 4D nonlinear betatronic motion in a particle accelerator can be conveniently described by using 4D symplectic mappings[9]. If we use canonical coordinates $\mathbf{x} = (x, p_x, y, p_y)$ to express the particle position at a given section of the circular machine, then the coordinates after one turn $\mathbf{x}' = (x', p_x', y', p_y')$ are given by

$$\mathbf{x}' = \mathbf{F}(\mathbf{x}) \qquad \mathbf{x} = (x, p_x, y, p_y). \tag{1}$$

The position of the particle after N turns around the machine is simply obtained by the repeated application of equation (1) N times. The linear motion is given by the direct product of two constant rotations in the planes (x, p_x) and (y, p_y) by the frequencies ν_x and ν_y also called the linear tunes of the machine.
To define the dynamic aperture, the first step is to consider the phase space volume of the initial conditions which are bounded after N iterations:

$$\int\int\int\int \chi(x, p_x, y, p_y)\, dx\, dp_x\, dy\, dp_y, \tag{2}$$

where $\chi(x, p_x, y, py)$ is the characteristic function of the set, i.e. it is unity if (x, p_x, y, p_y) is stable and zero otherwise. Since in 4D the invariant curves (i.e. 2D KAM tori) do not separate different domains of phase space, the concept of a last invariant curve surrounding stable initial conditions is not valid anymore[10, 9]. Therefore, from a purely theoretical point of view, the dynamic aperture could be a rather peculiar set, with holes and irregular structures. However, numerical simulations of lattices modelling circular machines showed that these situations are not typical[4, 11, 12, 13, 14]. This means that, in general, there exists a connected region of initial conditions which are stable for a given number of iterations.
As suggested in Ref. [2], two methods can be used to compute the integral in Eq. (2).

2.1 Method 1: direct integration

To exclude the disconnected part of the stability domain in the integral (2), we have to use polar variables $(r_1, \vartheta_1, r_2, \vartheta_2)$: r_1 and r_2 are the linear invariants. This choice is imposed by the fact that the linear motion is the direct product of rotations. As the nonlinear part of the equations of motion adds a coupling between the two planes, the perturbative parameter being the distance to the origin, it is quite natural to replace r_1

and r_2 with polar variables $r\cos\alpha$ and $r\sin\alpha$:

$$\begin{cases} x = r\cos\alpha\cos\vartheta_1 \\ p_x = r\cos\alpha\sin\vartheta_1 \\ y = r\sin\alpha\cos\vartheta_2 \\ p_y = r\sin\alpha\sin\vartheta_2; \end{cases} \qquad \begin{array}{r} r\in[0,+\infty[\\ \vartheta_1,\vartheta_2\in[0,2\pi[\\ \alpha\in[0,\pi/2] \end{array} \tag{3}$$

substituting in Eq. (2) we obtain

$$\int_0^{2\pi}\int_0^{2\pi}\int_0^{\pi/2}\int_0^{\infty}\chi(r,\alpha,\vartheta_1,\vartheta_2)\,r^3\sin(\alpha)\cos(\alpha)\,dr\,d\alpha\,d\vartheta_1\,d\vartheta_2. \tag{4}$$

Having fixed α, ϑ_1 and ϑ_2, let $r_{last}(\alpha,\vartheta_1,\vartheta_2)$ be the last value of r whose orbit is bounded after N iterations. Then, the volume of the stability domain is

$$A_{\alpha,\vartheta_1,\vartheta_2}=\frac{1}{8}\int_0^{2\pi}\int_0^{2\pi}\int_0^{\pi/2}[r_{last}(\alpha,\vartheta_1,\vartheta_2)]^4\sin(2\alpha)\,d\alpha\,d\vartheta_1\,d\vartheta_2. \tag{5}$$

In this way one excludes stable islands not connected to the main stable domain. The dynamic aperture will be defined as the radius of the hyper-sphere having the same volume as the stability domain:

$$r_{\alpha,\vartheta_1,\vartheta_2}=\left(\frac{2A_{\alpha,\vartheta_1,\vartheta_2}}{\pi^2}\right)^{1/4}. \tag{6}$$

To implement Eq. (5) as a computer code, one is forced to discretize the phase space variables $\vartheta_1,\vartheta_2,\alpha$ and r. If we consider a number of steps in the different variables given by $N_{\vartheta_1},N_{\vartheta_2},N_\alpha$ and N_r respectively, the dynamic aperture reads

$$r_{\alpha,\vartheta_1,\vartheta_2}=\left(\frac{\pi}{2N_\alpha N_{\vartheta_1}N_{\vartheta_2}}\sum_{i_\alpha=1}^{N_\alpha}\sum_{i_{\vartheta_1}=1}^{N_{\vartheta_1}}\sum_{i_{\vartheta_2}=1}^{N_{\vartheta_2}}[r_{last}(i_\alpha,i_{\vartheta_1},i_{\vartheta_2})]^4\sin(\frac{\pi i_\alpha}{N_\alpha})\right)^{1/4}, \tag{7}$$

where $r_{last}(i_\alpha,i_{\vartheta_1},i_{\vartheta_2})$ is defined over a discrete set of values.
The errors in the numerical evaluation of the dynamic aperture are mainly due to the discretization of the phase space variables, the relative errors being proportional to the inverse of the number of steps in the specified variable. Therefore, to reduce the total error one should choose integration steps which produce comparable errors, i.e. $N_{\vartheta_1}\propto N_{\vartheta_2}\propto N_\alpha\propto N_r$. In this way one can estimate the total relative error to be of the order of $1/N_r$ by computing $N\times N_r^4$ iterates. The fourth power in the number of orbits comes from the dimensionality of phase space, and makes a precise estimate of the dynamic aperture very CPU-time consuming.

2.2 Method 2: integration over the dynamics

To reduce the power in the dependence of the CPU-time on the number of steps N_r, it is possible to avoid the explicit scan on the variables (p_x,p_y) while taking indirectly into account their contribution to the values of the dynamic aperture. The algorithm proposed in Ref. [2] reads as follows:

- We fix $\overline{\vartheta}_1$ and $\overline{\vartheta}_2$. We scan over α and we find the radius $r(\alpha,\overline{\vartheta}_1,\overline{\vartheta}_2)$, computing the N iterates of the orbit.
- We divide $[0,2\pi[\times[0,2\pi[$ in M^2 equal squares (with $M^2 \leq N$), such that each square contains at least the phase p_x or p_y of one iterate of the last stable curve.
- For each square (m_1,m_2), where $m_1 = 1,\ldots,M$ and $m_2 = 1,\ldots,M$, we compute $r_{m_1,m_2}(\alpha,\overline{\vartheta}_1,\overline{\vartheta}_2)$, that is the average distance to the origin of the iterates which fall in that angular square.
- Finally, the dynamic aperture is computed as

$$r_{\alpha,d} = \left(\frac{\pi}{2N_\alpha} \sum_{i_\alpha=1}^{N_\alpha} [r_{ave}(i_\alpha)]^4 \sin(\frac{\pi i_\alpha}{N_\alpha}) \right)^{1/4} . \tag{8}$$

where

$$r_{ave}(i_\alpha) = \frac{1}{M^2} \sum_{m_1,m_2=1}^{M} r_{m_1,m_2}(i_\alpha,\overline{\vartheta}_1,\overline{\vartheta}_2) \tag{9}$$

In the previous definition we have to consider the average of r over each square (m_1,m_2) in order to take into account the fact that the variables p_x and p_y can have a rather irregular distribution along the orbit. This is a consequence of the presence of nonlinear resonances and hyperbolic fixed points.
In this case the error in the computation of the dynamic aperture is given by the discretization in the angles ϑ_1, ϑ_2 given by the M^2 squares over which the integration is carried out. The relative error in the dynamic aperture is proportional to $M^{-1} \propto N^{-1/2}$. Furthermore, the discretization of α and r introduce an additional source of error. Again, one should choose $N_\alpha \propto N_r \propto \sqrt{M}$. Therefore, to obtain a relative error of $1/N_r$ in the evaluation of the dynamic aperture, one has to evaluate only $N \times N_r^2$ iterates, thus saving a factor N_r^2 with respect to direct integration.

3 PARALLEL ALGORITHMS TO EVALUATE THE DA

In both algorithms to evaluate the DA, we can split the computations into two stages: in the first one the last stable radius, as a function of the phase space parameters, is determined, and then the final formula (7) or (8) is evaluated, corresponding to the computation of an average value of r. This suggests that a possible approach to implement a parallel algorithm would be to assign the initial conditions obtained by scanning over the phase space variables to different processors, in order to determine the stability of each initial condition. Then, once all the processors have finished, the results could be gathered to evaluate the sums (7) and (8). Unfortunately, this solution is far from being optimal and a more refined approach has to be found.

3.1 Parallel algorithm 1: direct approach

The direct method implies a scan over the four variables. The definition of the grid of initial conditions can be performed by using two do-loops: one over the ϑ_1, ϑ_2 and α, plus an inner loop over r:

Algorithm 3.1

```
1 $I_{max} := N_{\vartheta_1} * N_{\vartheta_2} * N_\alpha$
2 $I := 0$
```

```
$dynap := 0$
do while $I \leq I_{max}$
        $I_{\vartheta_1} := a(I, N_{\vartheta_1})$
        $I_{\vartheta_2} := b(I, I_{\vartheta_1}, N_{\vartheta_1}, N_{\vartheta_2})$
        $I_\alpha := c(I, I_{\vartheta_1}, N_{\vartheta_1}, I_{\vartheta_2}, N_{\vartheta_2})$ descriptors of 4D grid of initial conditions
        do while $I_r \leq N_r$
                $x := A(I_{\vartheta_1}, I_{\vartheta_2}, I_\alpha, I_r)$
                $p_x := B(I_{\vartheta_1}, I_{\vartheta_2}, I_\alpha, I_r)$
                $y := C(I_{\vartheta_1}, I_{\vartheta_2}, I_\alpha, I_r)$
                $p_y := D(I_{\vartheta_1}, I_{\vartheta_2}, I_\alpha, I_r)$                     initial coordinates
                $iflag := F(x, p_x, y, p_y;\ N)$        iteration of the initial condition
                if $iflag < 0$
                  then compute $r_{last}(I_{\vartheta_1}, I_{\vartheta_2}, I_\alpha)$ and quit the inner loop
                   else $I_r := I_r + 1$
                fi
        end
        $I := I + 1$
        $dynap := dynap + r^4_{last}(I_{\vartheta_1}, I_{\vartheta_2}, I_\alpha) * \sin(\frac{\pi I_\alpha}{N_\alpha})$
    end
    $dynap := (\pi * dynap / (2 * N_{\vartheta_1} * N_{\vartheta_2} * N_\alpha))^{1/4}$
```

In the algorithm, the procedures a, b, c are used to generate three indexes, to label the three independent variables ϑ_1, ϑ_2 and α, out of the single do-loop variable. A, B, C and D represent the coordinate transformation (3). Finally, F is the procedure used to iterate the initial condition N times including a stability test: the initial point is considered unstable if the distance of some iterates exceeds a given threshold. The result of the stability check is returned in the variable *iflag*.
The most efficient way to parallelise such a structure is to assign to each processor the task of determining the last stable radius for a given value of the other three variables, namely:

Algorithm 3.2

```
$I_{max} := N_{\vartheta_1} * N_{\vartheta_2} * N_\alpha$
$I := 0$
do while $I \leq I_{max}$
        $I_{\vartheta_1} := a(I, N_{\vartheta_1})$
        $I_{\vartheta_2} := b(I, I_{\vartheta_1}, N_{\vartheta_1}, N_{\vartheta_2})$
        $I_\alpha := c(I, I_{\vartheta_1}, N_{\vartheta_1}, I_{\vartheta_2}, N_{\vartheta_2})$ descriptors of 4D grid of initial conditions
        the $j-th$ processor with $I := balance(I, N_{proc}, j)$ now does
        do while $I_r \leq N_r$
                $x := A(I_{\vartheta_1}, I_{\vartheta_2}, I_\alpha, I_r)$
                $p_x := B(I_{\vartheta_1}, I_{\vartheta_2}, I_\alpha, I_r)$
                $y := C(I_{\vartheta_1}, I_{\vartheta_2}, I_\alpha, I_r)$
                $p_y := D(I_{\vartheta_1}, I_{\vartheta_2}, I_\alpha, I_r)$                     initial coordinates
                $iflag := F(x, p_x, y, p_y;\ N)$        iteration of the initial condition
                if $iflag < 0$
                  then compute $r_{last}(I_{\vartheta_1}, I_{\vartheta_2}, I_\alpha)$ then quit the inner loop
```

```
16                      else $I_r := I_r + 1$
17                  fi
18          end
19      end
20      the results from different processors are gathered together
21      to compute dynap
```

The parallel computations start at line 7: each processor performs the radial scan along a given direction in the 4D phase-space. The procedure *balance* determines which processor, among the group of N_{proc} units, will perform which subset of the cases of I. Once the first unstable initial condition is reached, the processor stops and it becomes available to compute along a different direction. In this way, one minimises the number of initial conditions which are actually considered with respect to the straightforward approach consisting of one processor per initial condition. In fact, in the case of K_r stable radial conditions for a given set of I_{ϑ_1}, I_{ϑ_2}, I_α, the proposed algorithm computes only $K_r + 1$ points to detect the border of stability. On the other hand, scanning along the radial variable using different processors implies that in the worst case one should consider $K_r + N_{proc}$. The parallel part of the algorithm ends at line 21: at this point the results obtained by the different processors are gathered together (one should keep in mind that each processor is unaware of the results obtained by the others) and only one CPU is used to perform the evaluation of the DA using Eq. (7).
As a final remark we would like to point out that typical values for $N_{\vartheta_1}, N_{\vartheta_2}, N_\alpha, N_r$ are of the order of 10. Therefore, in general, we have $N_{\vartheta_1} N_{\vartheta_2} N_\alpha \gg N_{proc}$ where N_{proc} represents the number of processors available and one can fully exploit the beneficial effects of parallelisation.

3.2 Parallel algorithm 2: integration over the dynamics

The integration over the dynamics allows us to compute the DA by scanning over two phase-space variables, namely α and r, and setting the angles ϑ_1, ϑ_2 to zero. In the original variables x, p_x, y, p_y the phases p_x, p_y are zero, while x and y vary according to the scan over r and α. The sequential algorithm is based on two nested loops: the outermost on α and the innermost on r to detect the last stable radius. The only difference with respect to the direct method is the evaluation of the average value $r_{m_1,m_2}(i_\alpha)$ once the border of stability has been reached:

Algorithm 3.3

```
1  $I_{max} := N_\alpha$
2  $I := 0$
3  dynap := 0
4  do while $I \le I_{max}$
5          $I_\alpha := c(I, N_\alpha)$              descriptor of 2D grid of initial conditions
6          do while $I_r \le N_r$
7                  $x := A(I_\alpha, I_r)$
8                  $p_x := 0$
9                  $y := C(I_\alpha, I_r)$
10                 $p_y := 0$                                        initial coordinates
11                 iflag := $F(x, p_x, y, p_y;\ N)$      iteration of the initial condition
12                 if iflag $< 0$
```

```
                  then compute $r_{ave}(I_\alpha)$ then quit the inner loop
                   else $I_r := I_r + 1$
              fi
        end
        $I := I + 1$
        $dynap := dynap + r^4_{ave}(I_\alpha) * \sin(\frac{\pi I_\alpha}{N_\alpha})$
    end
    $dynap := (\pi * dynap/\,(2 * N_\alpha N))^{1/4}$
```

As for the direct method, the most efficient way to parallelise this algorithm is to assign to each processor the task of determining the last stable radius as a function of α. Contrary to the previous case, the outer loop could have a length comparable to, or even smaller than, the maximum number of processors available. This means that the parameters N_α and N_{proc} cannot be determined independently each other. In some extreme situations, one can even consider a modification of the previous algorithm, in order to have one processor per initial condition: the disadvantages described in the previous section could be compensated by the possibility of evaluating the DA in a situation where $N_\alpha < N_{proc}$ without having any surplus processors.

3.3 Load balance

An important issue in the definition of a parallel algorithm is the load balance between the different processors; clearly, the optimal solution is to have an equal amount on work for each of the CPUs. The work-load is managed by the function $balance(I, N_{proc}, j)$ which controls the assignment of specific loop iterations to specific processors. Two principal methods of assigning work, static and dynamic, each with two variants, have been implemented and evaluated. The four different functions described below have been used:

- Static Block allocation: the values of the loop index I, over which we perform the parallelisation, are divided into blocks of length k, where $0 \leq k \leq [I_{max}/N_{proc}]$. The $j-th$ processor will work only on the cases with $I = j+i$, where $0 \leq i \leq k$. This means that the last processor will have a smaller number of cases than the others when $I \not\equiv j \bmod N_{proc}$. This approach is far from being optimal. In fact on top of this initial imbalance, one could have a non-uniform distribution of the work-load as a function of the loop index I. Then, the block allocation could generate a sort of systematic imbalance. For these reasons, this method was quickly abandoned in favour of cyclic allocation.
- Static Cyclic allocation: the values of the loop index I, over which we perform the parallelisation, are divided into blocks of length N_{proc}. The $j-th$ processor will only work on the cases with $I = j + kN_{proc}$, where $0 \leq k \leq [I_{max}/N_{proc}]$. Once again there would be an *ab initio* imbalance whenever $I \not\equiv j \bmod N_{proc}$. Although, in all cases where the CPU-time required to perform the scan along the radial variable varies considerably as a function of the variable I, the efficiency of the parallel algorithm will be reduced as well, this approach works well for situations in which the work-load is already well balanced from the beginning. This approach is also rather appealing due to its simplicity and, furthermore, it exploits all the processors available for the computation (except possibly the last).
- Dynamic allocation with a Master: this technique allows very high computational efficiency even in those cases where the difference in CPU-time needed for each loop iteration is significant. As soon as a processor is available (i.e. has finished

a previous loop iteration), it will start the calculations on the case labelled by the current value of the index I. The structure of the procedure $balance(I, N_{proc}, j)$ is much more complex than in the previous case and it is based on a **master-slave** structure. A processor is defined to be the master: it does not take part in the actual computations, but it dialogues with the other units. As soon as a CPU has finished its task, it sends a message to the master. At this point the master process assigns a new task to the slave, keeping track of the remaining cases. It is clear that in this way the use of processing units is optimised. The drawbacks are the fixed overhead of $1/N_{proc}$, the overhead due to the communications between the master and the slaves, and a possible bottleneck if many slaves request a new iteration at the same time. In the cases considered, with N_{proc} typically between 20 and 80, these effects were negligible with respect to the improvement introduced by the more optimal load-balance.

- **Atomic Dynamic allocation**: this technique provides the same advantage as dynamic load balancing as described above, but without the fixed overhead of a master processor. Every processor uses a global shared variable I as the loop index. Each processor simply performs an atomic 'read and add one' to the index, guaranteeing the uniqueness of the value assigned to each processor. As in the previous cases, the results from each processor are accumulated, allowing further computation and output of results over the entire computational domain. This method is also extremely simple, but can be implemented only on systems with a global shared memory capability such as the MEIKO CS-2.

All of the above techniques provide a significant reduction in the real time to solution of the problem, as compared to the sequential version of the program. Further improvements in load balancing would require prior knowledge of the computational requirement of each loop iteration.

3.4 Implementation Considerations

The main issue was a choice of parallelisation methods, given a problem with considerable natural parallelism (many independent cases to be studied), and a large existing 35,000 line Fortran sequential program (PLATO[3]). This is a common situation, with a further common constraint on the effort available. It was thus important to minimise the changes to the existing program, and to ensure portability across the wide range of parallel systems available today. Once the algorithms described above had been agreed, which minimised communication and implied minimal source code changes, there remained a choice of implementation tools.

To have relied on a parallel processing system with shared memory would have considerably restricted the portability and scalability of the application; furthermore, methods of parallelisation for such systems tend to be system specific. While some consideration was given to the possibility of using a High Performance Fortran (HPF) compiler such a tool is not always available and does not typically handle load imbalance well. Thus although, in principle, we were looking at data parallel problem, in the sense of dividing up a computational domain, we preferred to use a message passing tool given the limited amount of communication necessary for each loop iteration.

The choice here was basically between the PVM and MPI message passing systems. Both are widely available, public-domain implementations are freely available, and both provide the necessary functionality. In the end MPI was chosen because of the excellent implementations available, because of successful previous experience, and because the dynamic process creation facilities of PVM were not required.

Subsequently, given the ease of implementation of the chosen algorithms, a MEIKO specific solution was developed, to demonstrate the effectiveness and simplicity of a logically shared memory system.

- **MPI library**: the Message Passing Interface[15, 16, 17] is a portable library designed for parallel applications and for portability. It provides a wide range of functionality of which only a subset was required. In particular, it was necessary for each process to determine the total number of processes active, and its own rank within this set. Otherwise only simple synchronous SEND, RECEIVE, and GATHER message functions were used apart from a few instances where global REDUCE functions for SUM, MAX and MIN proved of some utility. Total changes and additions to the original program amounted to only a few hundred lines.
- **Atomic library**: the Atomic library is a MEIKO specific library. Therefore the code based on this library cannot be ported directly to other platforms. On the other hand use of this tool provided the most efficient and simplest solution to our problem. Conceptually, this tool is easy to understand, given the inherent difficulty in thinking parallel, provides a simple synchronisation/locking capability and an easy way to exchange information. Unlike the message passing libraries which are extremely rich and try to provide specific procedures for a wide range of application specific problems, the atomic library provides some simple primitives which can be used to build up more complicated procedures.
  The basic functions are,
  - at_sync: wait until all processes have reached this point.
  - at_add(destination, address, increment) increment the contents of address on process destination by value. Returns the old value (before the increment).
  - at_readwrite(destination, address, increment) swap the contents of address on process destination with value. Returns the previous value.
  - at_testwrite(destination, address, test, value) test the contents of address on process destination and if they are equal to test replace by value. Returns the previous value.

  Given that addresses are identical in all processes, since multiple copies of the same process are used everywhere, these simple tools are extremely effective.

  In fact only the first three functions were required, demonstrating the simplicity and effectiveness of this design. By default all variables are private to a process. This facilitates debugging and gives the programmer an awareness of where communication is being performed (as in a message passing system). However it obviates the need for much explicit synchronisation, maintains symmetry, and also obviates the need for a master process in many task farming applications.

4 MEIKO DESCRIPTION

The MEIKO CS-2 computer is a distributed-memory, scalable, parallel system using SPARC micro-processors and a MEIKO-developed interconnect which enables programs to read and write memory in remote nodes without context switching. The CERN CS-2 has 64 nodes, each with two 100 MHz HyperSPARC processors (rated at over 100 Specint92 per processor) and 128 MB of memory. Each node has a local disk for temporary data storage and paging or swapping as well as SCSI connections for additional peripheral equipment. Each node maintains its own copy of the Solaris operating system kernel although the system routines and utilities may be down-loaded from dedicated server nodes. The entire system could be considered simply as a cluster of workstations were it not for the high speed network and MEIKO developed software.

Each node in the system is inter-connected using its own Elan processor and an Elite scalable network. The network is expanded to provide a 50MB/second bandwidth between nodes even if the number of nodes increases. The latency of communication is extremely low, ranging from less than 10 micro-seconds using MEIKO communication primitives, to less than a 100 micro-seconds when using high-level libraries like MPI.

MEIKO extensions to the standard Solaris Operating System provide for effective and flexible use of the machine although other common features such as NFS/AFS file systems, tcp/ip protocols, and the Network Queueing System (NQS) are also fully supported, as are the TotalView parallel debugger and a wide range of SPARC compilers. The CS-2

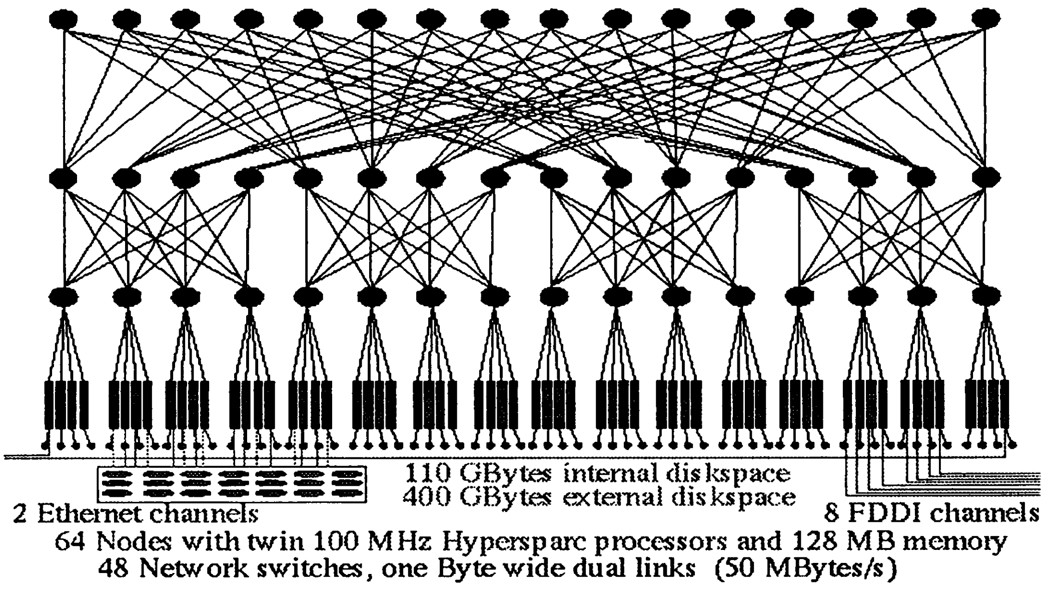
110 GBytes internal diskspace
400 GBytes external diskspace
2 Ethernet channels
8 FDDI channels
64 Nodes with twin 100 MHz Hypersparc processors and 128 MB memory
48 Network switches, one Byte wide dual links (50 MBytes/s)

Figure 1: Schematic view of the CS-2 structure. The connections between the different processors are shown.

service is at present used for the support of high speed parallel central data recording using multiple network, disk, and tape streams, event reconstruction for high energy physics experiments, and for event-parallel simulation using a specially developed version of the GEANT program[18]. It also provides a Parallel Interactive Analysis Facility (PIAF[19]), a highly data parallel input/output intensive application. As the development and deployment of the system was extensively supported by the European Union a significant number of industrial codes have also been ported and bench-marked on the CERN CS-2. Currently under development is a system (GRACE[20, 21]) for the automatic generation of Feynman diagrams in parallel.

5 RESULTS

The different algorithms presented in Section 3 have been tested using a realistic model of the CERN Super Proton Synchrotron (SPS). The machine has been used in the past years to perform some experiments in the framework of the Large Hadron Collider (LHC) project. To understand the beam dynamics in conditions similar to those foreseen for a superconducting machine dominated by strong nonlinear magnetic errors, the behaviour of the dynamic aperture has been studied[22, 23, 24]. Under normal operational conditions, the machine behaves very linearly, but eight strong sextupoles (on top of the 108 sextupoles used to correct the linear chromaticity), normally used for the resonant extraction of the beam, allow us to introduce in a controlled way a nonlinear perturbation. In the configuration used during the experimental sessions[24], the extraction sextupoles were powered in such a way to introduce strong nonlinear effects. Furthermore, the linear tunes were set to the values $\nu_x = 26.637, \nu_y = 26.533$ near a resonance of order 7 to test the combined effect of resonances and nonlinearities on the beam stability.

As a first step, we have analysed the performance of the algorithm 3.2 using the model of the SPS lattice to compute the dynamic aperture. For such computations, we have used $N_{\vartheta_1} = N_{\vartheta_2} = N_\alpha = 10$, while the number of iterations N has been fixed to 1000. As far as the number of radial initial conditions is concerned, we have specified the step between two successive points: the program increases the value of the radial variable until an unstable condition is met. Therefore N_r is not fixed. This is very convenient whenever an upper bound to the value of r_{last} is not known a priori. The simulations have been used to test how the CPU-time needed to compute the DA scales as a function of N_{proc}, independently of the different type of implementation. The results are shown in Fig. 2. Independently of the implementation, the CPU-time decreases exponentially with

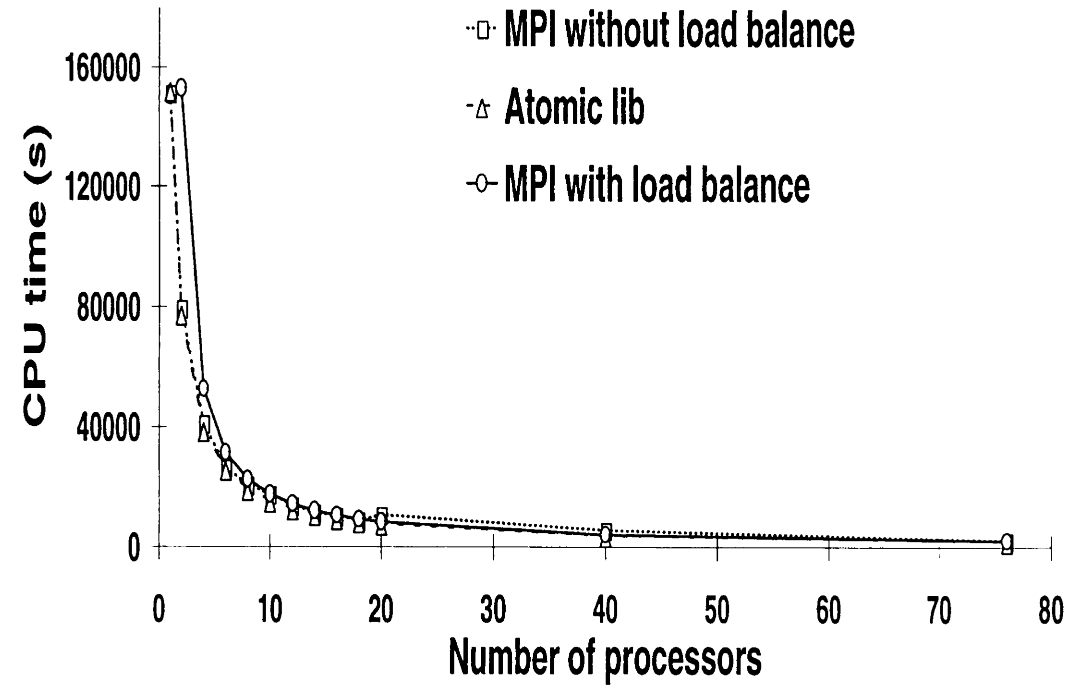

Figure 2: Performances of the first parallel algorithm. The total CPU-time is depicted as a function of N_{proc}.

the number of processors: every time one increases N_{proc} by a factor two, the CPU-time

decreases by approximately the same factor. In this respect the performance seems to be optimal. As expected, using the MPI library has some drawbacks: one processor is used as a master, therefore the total number of active CPUs is reduced by one with respect to the other methods. This effect is clearly visible on the plot. As soon as the N_{proc} exceeds 10 units, this negative effect disappears and the beneficial effect of a good load balance makes this particular implementation more efficient than using MPI alone without load balance and practically at the same level as the algorithm based on Atomic library.
To quantify the impact of the load balance approach on the efficiency of the algorithm, we plot the normalised mean CPU time, namely:

$$\mu(N_{proc}) = \frac{1}{N_{proc}} \sum_{i=1}^{N_{proc}} \frac{\tau_i}{\tau_{max}}, \tag{10}$$

where $\tau_{max} = \max_{j=1,\cdots N_{proc}} \tau_i$, and τ_i represents the total CPU-time used by the $i-th$ processor. In the ideal case of perfect load balance $\mu(N_{proc}) = 1$. We also use the standard deviation to measure the dispersion of the τ_i around the mean value. The results are reported in Fig. 3. The difference in performance between the three implementations is

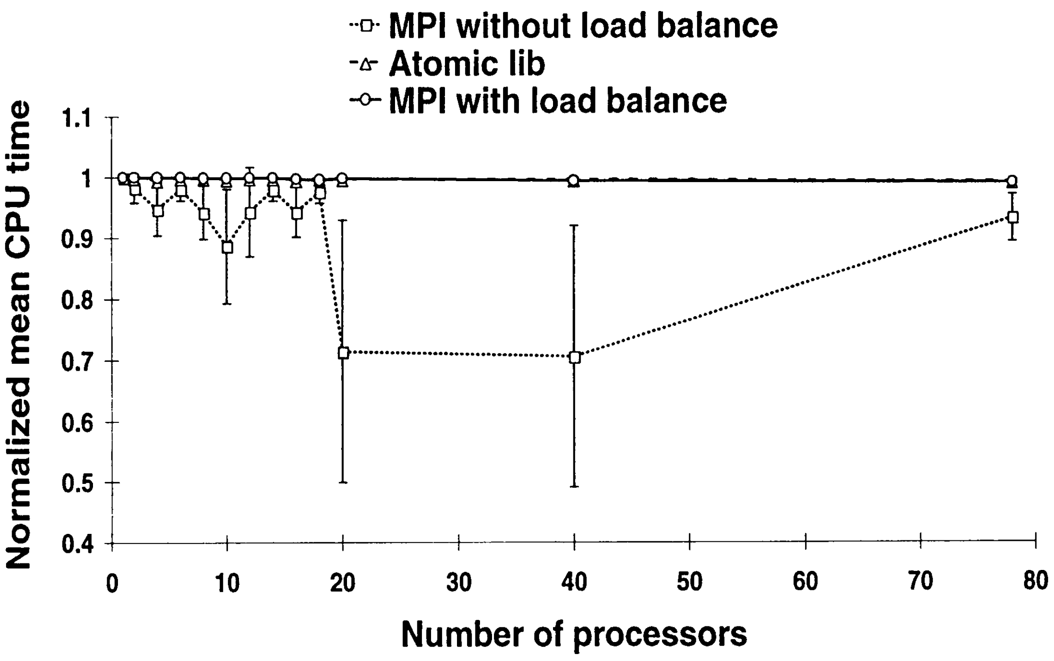

Figure 3: Load balance for the first parallel algorithm. The average CPU-time μ is shown as a function of N_{proc}. The error bars are computed using the standard deviation of the CPU-time for the different processors.

clearly seen. For the algorithm without any built-in load balance control, the results are rather poor: the normalised mean time fluctuates wildly, especially when the number of processors is greater than 20. Correspondingly the CPU-time performance (see Fig. 2) becomes worse than the other algorithms.
On the other hand the other two approaches behave very well: in both cases μ is almost constant as a function of N_{proc} and very close to one. In this respect the implementation based on MPI and the one using the Atomic library are almost equivalent, because the time spent in communications is negligible with respect to the CPU-time required for

computations.
The same kind of tests have been carried out also for the algorithm 3.3. The results do not differ significantly from the ones previously shown. However, for this specific case we have also implemented a parallel algorithm in which the parallelisation is performed not only on the radial variable, but on both parameters α and r defining the 2D grid of initial conditions. The results are shown in Fig. 4. It is apparent the enormous difference

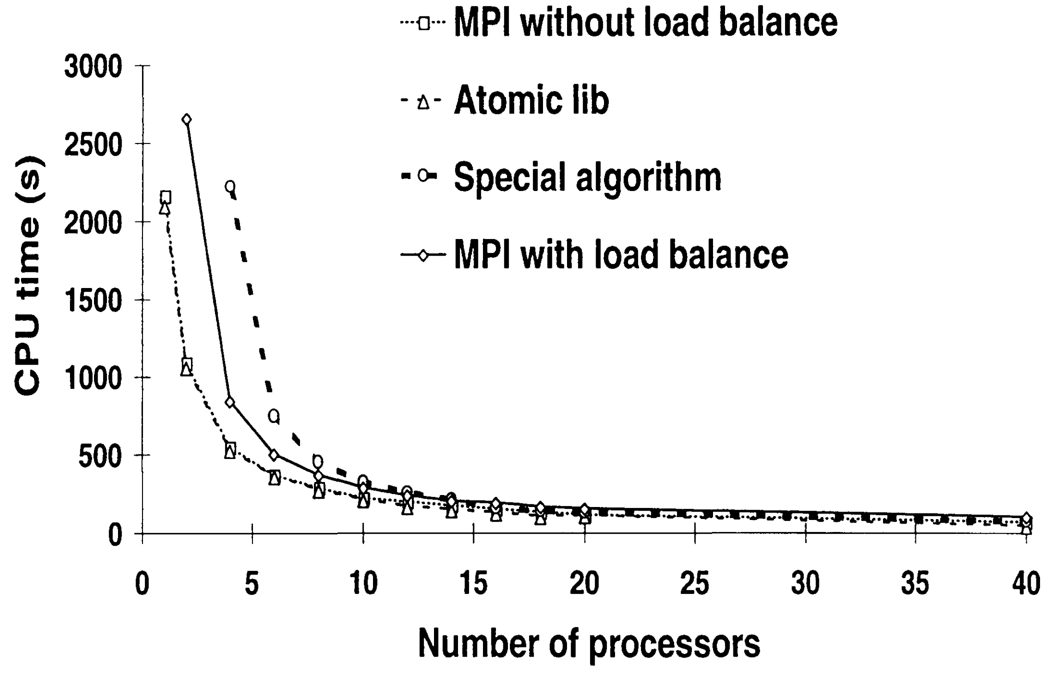

Figure 4: Performances of the second parallel algorithm. The total CPU-time is depicted as a function of N_{proc}. We have also shown the results obtained by parallelising over the whole 2D grid of initial conditions instead of the radial variable only.

between the CPU-time for algorithm 3.2 and algorithm 3.3 (almost a factor 50). This reflects the fact that algorithm 3.2 performs a scan over the four phase space variables, while algorithm 3.3 varies two variables only.
In this case we have used $N_{alpha} = 100$, and the number of iterations is again set to 1000. Also in this case N_r is not fixed, instead we impose a value for the step size along the radial variable. The special algorithm is less efficient than the others: this is a consequence of the over-computations discussed in Section 3.

6 CONCLUSIONS

In this paper we have reviewed two sequential algorithms to compute the dynamic aperture in the presence of strong nonlinear effects. The original part consists in the development of analogous techniques based on parallel algorithms. After a detailed discussion on the algorithmic structure, on the load balance and on the implementation using different libraries, we have studied the performances of the algorithms using a realistic model of the CERN SPS. The results show that the algorithms are optimal and that the problem of load balance can be efficiently solved. The solutions based on the MPI and Atomic library have very similar performance: although the latter gives CPU-time shorter by some percent than the version implemented with MPI, the overall results are comparable.

The positive results obtained so far encourage us to develop new algorithms in view of applications to real problems in the field of particle accelerators. The next step will be to design a parallel approach to the problem of sorting, in order to find an optimal solution in a short time. Another possible application of these techniques could be the analysis of collective effects, involving the simulation of a large number of interacting particles.

7 Aknowledgements

We would like to express our gratitude to M. Metcalf, D. Mohl and D. Warner for carefully reading the manuscript.

References

[1] The LHC study group, CERN AC (LHC) **95–05** (1995).

[2] E. Todesco and M. Giovannozzi, Phys. Rev. **53**, 4067 (1996).

[3] M. Giovannozzi, E. Todesco, A. Bazzani and R. Bartolini, CERN PS (PA) **96–12** (1996).

[4] M. Giovannozzi, R. Grassi, W. Scandale and E. Todesco, Phys. Rev. E **52**, 3093 (1995).

[5] R. Bartolini, M. Giovannozzi, W. Scandale and E. Todesco, CERN LHC PROJECT REPORT **96–38** (1996).

[6] R. D. Ryne, S. Habib and T. P. Wangler, in *Sixteenth Biennial Particle Accelerator Conference*, edited by LANL et al. (IEEE Operations Center, Piscataway, NJ, 1996) pp. 3149.

[7] J-M. Lagniel and D. Libault, in *Sixteenth Biennial Particle Accelerator Conference*, edited by LANL et al. (IEEE Operations Center, Piscataway, NJ, 1996) pp. 3235.

[8] G. F. Dell and S. Peggs, in *Sixteenth Biennial Particle Accelerator Conference*, edited by LANL et al. (IEEE Operations Center, Piscataway, NJ, 1996) pp. 2327.

[9] A. Bazzani, E. Todesco, G. Turchetti and G. Servizi, CERN **94–02** (1994).

[10] J. D. Meiss, Rev. Mod. Phys. **64**, 795 (1992).

[11] W. Scandale, in *Third European Particle Accelerator Conference* , edited by H. Henke (Edition Frontiéres, Gif sur Yvette, 1993) pp. 264.

[12] F. Zimmermann, in *Fourth European Particle Accelerator Conference* , edited by V. Sueller et al. (World Scientific, Singapore, 1995) pp. 327.

[13] F. Galluccio and W. Scandale, CERN SL (AP) **89-51** (1989).

[14] F. Galluccio and F. Schmidt, in *Third European Particle Accelerator Conference*, edited by H. Henke (Edition Frontiéres, Gif sur Yvette, 1993) pp. 640.

[15] W. Gropp, E. Lusk and A. Skjellum, Using MPI (MIT Press, Cambridge, 1994).

[16] I. Foster, Designing and Building Parallel Programs (Addison-Wesley, Reading, 1995).

[17] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, MPI: The Complete Reference (The MIT Press, Cambridge, 1995).

[18] Application Software Group, CERN Program Library Long Writeups **Q123**.

[19] T. Hakulinen and F. Rademakers, to be published in the Proceedings of *HPCN 95, High Performance Computing and Networking Conference*, Milan - Italy, 2-5 May, 1995.

[20] T. Kaneko, Comput. Phys. Comm. **92**, 127 (1995).

[21] F. Yuasa, D. Perret-Gallix, S. Kawabata and T. Ishikawa, to be published in the Proceedings of *Fifth International Workshop on Software Engineering, Artificial In-*

telligence and Expert Systems for High Energy and Nuclear Physics, Lausanne - Switzerland, 2-5 Sep., 1996.

[22] X. Altuna et al., in *Advanced ICFA Beam Dynamics Workshop*, edited by A. W. Chao (AIP Conf. Proc. No. 255, New York, 1992) pp. 355.

[23] J. Gareyte, W. Scandale, and F. Schmidt, in *International Workshop on Nonlinear Problems in Accelerator Physics*, edited by M. Berz et al. (Institute of Physics, Bristol, 1993) pp. 235.

[24] W. Fischer, M. Giovannozzi and F. Schmidt, CERN SL (AP) **95–96** (1995).