Stochastic Simulation of a Marine Host-Parasite System Using a Hybrid MPI/OpenMP Programming

Michel Langlais[1,⋆], Guillaume Latu[2,*], Jean Roman[2,*], and Patrick Silan[3]

[1] MAB, UMR CNRS 5466, Université Bordeaux 2,
146 Léo Saignat, 33076 Bordeaux Cedex, France
Michel.Langlais@sm.u-bordeaux2.fr
[2] LaBRI, UMR CNRS 5800, Université Bordeaux 1 & ENSEIRB
351, cours de la Libération, 33405 Talence, France
{latu|roman}@labri.fr
[3] UMR CNRS 5000, Université Montpellier II, Station Méditerranéenne de l'Environnement Littoral, 1 Quai de la Daurade, 34200 Sète, France
silan@univ-montp2.fr

Abstract. We are interested in a host-parasite system occuring in fish farms, *i.e.* the sea bass - *Diplectanum aequans* system. A discrete mathematical model is used to describe the dynamics of both populations. A deterministic numerical simulator and, lately, a stochastic simulator were developed to study this biological system. Parallelization is required because execution times are too long. The Monte Carlo algorithm of the stochastic simulator and its three levels of parallelism are described. Analysis and performances, up to 256 processors, of a hybrid MPI/OpenMP code are then presented for a cluster of SMP nodes. Qualitative results are given for the host-parasite system.

1 Introduction

Host-parasite systems can present very complex behaviors and can be difficult to analyse from a purely mathematical point of view [12]. Ecological and epidemiologic interests are motivating the study of their population dynamics. A deterministic mathematical model (using some stochastic elements) for the sea bass–*Diplectanum aequans* system was introduced in [3,6]. It concerns a pathological problem in fish farming. Numerical simulations and subsequent quantitative analysis of the results can be done, and a validation of the underlying model is expected. Our first goal in this work is to discover the hierarchy of various mechanisms involved in this host-parasite system. A second one is to understand the sensitivity of the model with respect to the initial conditions. In our model, many factors are taken into account to accurately simulate the model, *e.g.* spatial and temporal heterogeneities. Therefore, the realistic deterministic

⋆ Research action ScAlApplix supported by INRIA.

B. Monien and R. Feldmann (Eds.): Euro-Par 2002, LNCS 2400, pp. 436–446.

simulator has a significant computation cost. Parallelization is required because execution times of the simulations are too long [7].

Individual-Based Models (IBM) are becoming more and more useful to describe biological systems. Interactions between individuals are simple and local, yet can lead to complex patterns at a global scale. The principle is to replicate several times the simulation program to obtain statistically meaningful results. In fact, a single simulation run driven by a sequence of pseudo-random numbers is not representative for a set of input parameters. Then, outputs are averaged for all theses *simulation runs* (or *replicates*). The Individual-Based Model approach contrasts with a more aggregate population modeling approach, and provides a mechanistic rather than a descriptive approach to modeling. Stochastic simulations reproduce elementary processes and often lead to prohibitive computations. Hence, parallel machines were used to model complex systems [1,8,9].

In this work, a description of the biological background and of performances of the deterministic simulator is briefly given. Next, we present the main issues concerning the parallel stochastic simulator. We point out the complexity of computations and, then, we develop our parallel algorithmic solution and investigate its performances. Hybrid MPI and OpenMP programming is used to achieve nested parallelization. Finally, we present some of the biological results obtained for an effective implementation on a SP3 IBM machine. This work received a grant from ACI bio-informatique. This project is a collaborative effort in an interdisciplinary approach: population dynamics with CNRS, mathematics with Université Bordeaux 2, computer science with Université Bordeaux 1.

2 Description of the Biological Background

In previous works [3,6,12], the mathematical model of the host-parasite system was presented; a summary is given now. The numerical simulation is mainly intended to describe the evolution of two populations, hosts and parasites, over one year in a fish farm. After a few time steps, any parasite egg surviving natural death becomes a larva. A time step $\Delta t = 2$ days corresponds to the average life span of a larva. The larva population is supplied by eggs hatching and by an external supply (larvae coming from open sea by pipes). An amount of $L(t)$ larvae is recruited by hosts, while others die. Highly parasitized hosts tend to recruit more parasites than others do. This means that the parasite population is overdispersed or aggregated with the host population. Most parasites are located on a few hosts. A detailed age structure of the parasites on a given host is required because only adult parasites lay eggs, while both juvenile and adult parasites have a negative impact on the survival rate of hosts. The population of parasites is divided into $K = 10$ age classes, with 9 classes of juvenile parasites and one large class for adult parasites. We consider that only a surfeit of parasites can lead to the death of a host. Environmental and biological conditions are actually used in the simulations, *e.g.* water temperature $T(t)$, death rate of parasites $\mu(T(t))$. The final goal is to obtain values of state variables at each time step.

3 Deterministic Numerical Simulation

The elementary events of one time step are quantified into probabilistic functions describing interactions between eggs, larvae, parasites and hosts. The frequency distribution of parasite numbers per host is updated with deterministic equations (without random number generation). Let $C(K, S)$ be the complexity of one *time step*. The S variable is limited to the minimum number of parasites that is lethal for a fish (currently $S \leq 800$); K is the number of age classes used ($K = 10$). A previous study [5] led to a reduced update cost of $C(K, S) = K\,S^4$ for one *time step* Δt, and one has $C(10, 800) = 950$ GFLOP. This large cost comes from the fine distribution of parasites within the host population, taking care of the age structure of parasites. A matrix formulation of the algorithm allows us to use BLAS 3 subroutines intensively, and leads to large speedups. Different mappings of data and computations have been investigated. A complete and costly simulation of 100 TFLOP lasts only 28 minutes on 128 processors (IBM SP3 / 16-way NH2 SMP nodes of the CINES[1]) and 9 minutes on 448 processors. The performance analysis has established the efficiency and the scalability of the parallel algorithm [7]. Relative efficiency of a 100 TFLOP simulation reached 83% using 128 processors and 75% using 448 processors.

4 Stochastic Model of Host-Parasite Interactions

For an Individual-Based Model, basic interactions are usually described between the actors of the system. Hosts and settled parasites are represented individually in the system, while eggs and larvae are considered globally. This allows to compare the deterministic and stochastic simulators, because only the inter-relationship between host and parasite populations are modeled differently. The deterministic simulator produces one output for a set of input parameters, whereas the stochastic simulator needs the synthesis of multiple different simulation runs to give a representative result. The number of replications R will depend on the desired accuracy of the outputs. We now describe how to manage host-parasite interactions. Let $\mathbb{H}_i$ be the host object indexed by i. Let $\mathbb{H}_i^p$ be the amount of parasites on the host $\mathbb{H}_i$.

- The probability for the host $\mathbb{H}_i$ to die, between time t and $t+\Delta t$, is given by $\pi(\mathbb{H}_i^p)$. A random number x is uniformly generated on $[0, 1]$ at each time step and for each living host $\mathbb{H}_i$. If $x \leq \pi(\mathbb{H}_i^p)$, then $\mathbb{H}_i$ dies.
- Consider that $\mathbb{P}_i(q)$ is the amount of parasites of age $q\Delta t$ settled on the host $\mathbb{H}_i$. Assume that the water temperature is $T(t)$, the death rate of parasites is $\mu(T(t))$. A binomial distribution $\mathcal{B}(\mathbb{P}_i(q); \mu(T(t)))$ is used to compute how many parasites among $\mathbb{P}_i(q)$ are dying during the time step t. All surviving parasites are moved to $\mathbb{P}_i(q+1)$ (for $0 < q < K$), see figure 1. In short, for each host and each age class of parasites, a random number is generated using a binomial distribution to perform the aging process of parasites.

[1] Centre Informatique National de l'enseignement supérieur - Montpellier, France.

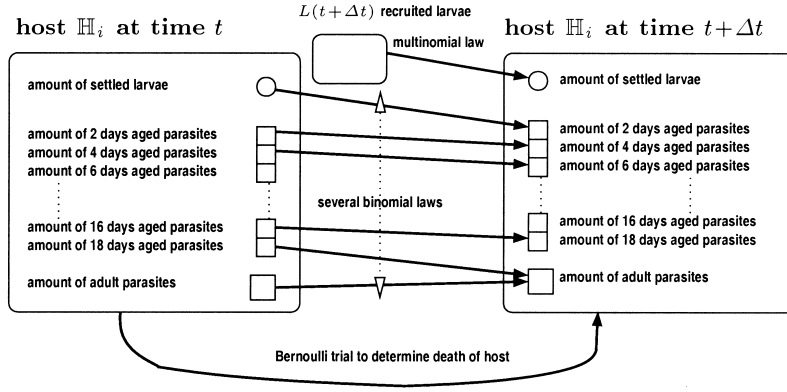

Fig. 1. Update of all living hosts and parasites at time t

- A function $f(p,t)$ gives the average percentage of larvae that are going to settle on a host having p parasites. Let $L(t)$ be the number of recruited larvae, one has:

$$\sum_{i/with\ \mathbb{H}_i\ living\ at\ time\ t} f(\mathbb{H}_i^p, t)\, L(t) = L(t)\,. \tag{1}$$

The recruitment of $L(t)$ larvae on $H(t)$ hosts must be managed. Each host $\mathbb{H}_i$ recruits a larva with mean $f(\mathbb{H}_i^p, t)$. Let R_i be the variable giving the number of larvae recruited by $\mathbb{H}_i$ at time $t+\Delta t$. Let $i_1, i_2.., i_{H(t)}$ be the indices of living hosts at time t. To model this process, a multinomial distribution is used: $(R_{i_1}, R_{i_2}, ..R_{i_{H(t)}})$ follows the multinomial distribution $\mathcal{B}(L(t);\ f(\mathbb{H}_{i_1}^p, t),\ f(\mathbb{H}_{i_2}^p, t).., f(\mathbb{H}_{i_{H(t)}}^p, t))$. One has the property that $R_{i_1}+R_{i_2}+..+R_{i_{H(t)}} = L(t)$.

5 Stochastic Algorithm

The algorithm used in the stochastic model is detailed in figure 2. Parts related to direct interactions between hosts and parasites (*i.e.* 2.2.6, 2.2.7 and 2.2.8) represent the costly part of the algorithm.

On a set of benchmarks, these correspond to at least 89% of execution time for all simulation runs. For simulations with long execution times, parasites and hosts appear in large numbers. For this kind of simulations, the epizooty develops for six months. One can observe more than 4×10^3 hosts and 10^6 parasites at a single time step. The most time consuming part of this problem is the calculation of the distribution of larvae among the host population (2.2.6 part). With the elementary method to reproduce a multinomial law, it means a random trial per recruited larva; the complexity is then $\Theta(L(t))$. In the 2.2.7 part, the number of Bernoulli trials to establish the death of hosts corresponds to a complexity $\Theta(H(t))$. In the 2.2.8 part, each age class $q\Delta t$ of parasites of each

```
1.  read input parameters;
2.  For all simulation runs required r ∈ [1, R] ;
    2.1  initialize, compute initial values of data;
    2.2  for t := 0 to 366 with a time step of 2
         2.2.1  updating environmental data;
         2.2.2  lay of eggs by adult parasites;
         2.2.3  updating the egg population (aging);
         2.2.4  hatching of eggs (giving swimming larvae);
         2.2.5  updating the larva population (aging);
         2.2.6  recruitment of larvae by hosts;
         2.2.7  death of over-parasitized hosts;
         2.2.8  updating the parasite population on hosts (aging);
         End for
    2.3  saving relevant data of simulation run ''r'';
    End for
3.  merging and printing results of all simulation runs.
```

Fig. 2. Global algorithm

host i is considered to determine the death of parasites. For each and every one, one binomial trial $\mathcal{B}(\mathbb{P}_i(q); \mu(T(t)))$ is done, giving a $\Theta(K \times H(t))$ complexity. So, one time step of one simulation run grows as $\Theta(H(t) + L(t))$. For a long simulation, the 2.2.6 part can take up to 90 % of the global simulation execution time, and after a few time steps, one has $H(t) \ll L(t)$. In that case, the overall complexity of the simulation is $\Theta(\sum_{t\in[0,366]} L(t))$. The sum of recruited larvae over one year reaches 2×10^8 in some simulations. Considering R replications, the complexity is then $R\ \Theta(\sum_{t\in[0,366]} L(t))$.

The main data used in the stochastic simulator are hosts and age classes of parasites. The memory space taken for these structures is relatively small in our simulations: $\Theta(K\,H(t))$. Nevertheless, to keep information about each time step, state variables are saved to do statistics. For J saved variables and 183 steps, the space required for this record is $\Theta(183\,J\,R)$, for all simulation runs.

6 Multilevel Parallelism for Stochastic Simulations

Several strategies of parallelization are found in the literature for stochastic simulations. First, all available processors could be used to compute one simulation run; simulation runs are then performed one after the other. Generally, a spatial decomposition is carried out. In multi-agent systems, the space domain of agent interactions is distributed over processors [8,9]. For a cellular automaton based algorithm, the lattice is split among processors [1]. Nevertheless, this partitioning technique is available only if the granularity of computation is large enough, depending on the target parallel machine.

A more general approach for a stochastic simulation consists in mapping replicates onto different processors. Then, totally independent sequences of instructions are executed. At the end of all simulation runs, outputs are merged to generate a synthesis, *i.e.* means and standard deviations of state variables for each time step. However, this approach shows limitations. If simulation runs have not equal execution times, it leads to load imbalance. This potential penalty could be

partly solved with dynamic load balancing, if simulation runs could be mapped onto idle processors, whenever possible. The required number of simulation runs is a limitation too, because one has for P processors, $P \leq R$. Finally, the overhead of the step used to generate final outputs must be significantly lower than the cost of simulation runs. This second approach is often described [2], because it leads to massive parallelization. The problem remains of generating uncorrelated and reproducible sequences of random numbers on processors.
Finally, the validation of simulation models may require a sensitivity analysis. Sensitivity analysis consists in assessing how the variation in the output of a model can be apportioned, qualitatively or quantitatively, to different sources of variation in the input. It provides an understanding of how the output variables respond to changes in the input variables, and how to calibrate the data used. Exploration of input space may require a considerable amount of time, and may be difficult to perform in practice. Aggregation and structuring of results consume time and disk space. Now, a sequence of simulations using different input sets could be automated and parallelized. The synthesis of final outputs need the cooperation of all processors. This third level of parallelism is described in [4], however often unreachable for costly simulations. As far as we know, no example of combining these different levels of parallelism appears in the literature.

7 Parallel Algorithm

Most recent parallel architectures contain a large number of SMP nodes connected by a fast network. The hybrid programming paradigm combines two layers of parallelism: implementing OpenMP [11] shared-memory codes within each SMP node, while using MPI between them. This mixed programming method allows codes to potentially benefit from loop-level parallelism and from coarse-grained parallelism. Hybrid codes may also benefit from applications that are well-suited to take advantage of shared-memory algorithms. We shall evaluate the three levels of parallelism described above within the framework of such SMP clusters. Our parallel algorithm is presented in figure 3.

At the first level of parallelism, the process of larvae recruitment can be distributed (2.2.6 part of the algorithm). A sequence of random numbers is generated, then the loop considering each larva is split among the processors. Each OpenMP thread performs an independent computation on a set of larvae. This fine-grain parallelism is well suited for a shared-memory execution, avoiding data redundancy and communication latencies.

Suppose we do not use the first level of parallelism; the second level of parallelism means to map simulation runs onto the parallel machine. Typically, each processor gets several simulation runs, and potentially there is a problem of load imbalance. However, benchmarks have established that execution times of simulation runs do not have large variability for a given set of input parameters of a costly simulation. So, if each processor has the same number of replicates to carry out, the load is balanced. MPI is used to perform communications. In fact, the use of OpenMP is not a valuable choice here, because it prevents the

```
For all simulations a ∈ [1, A] of the sensitivity analysis do in // {
.  read input parameters;
.  For all simulations runs r ∈ [1, R] do in //
.  .  compute initial values of state variables;
.  .  For t:=0 to 366 with a time step of 2 do
.  .  .  update of steps 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5;
.  .  .  parallel update of step 2.2.6; (* OpenMP threads *)
.  .  .  update of steps 2.2.7, 2.2.8;
.  .  }
.  }
.  gather outputs of simulation a (MPI collective communication);
}
print outputs;
```

Fig. 3. Parallel algorithm

execution on several SMP nodes. When all simulation runs are finished, a gather step is performed with a MPI global communication routine.

When performing a sensitivity analysis (third level of parallelism), the external loop (a variable) is distributed among sp sets of processors. Each set has m processors, so the total number of processors is $sp \times m = P$. The values of the a incices are assigned to the sp sets in a cyclic manner to balance the load. Next, the values of the r indices are mapped onto m processors. To get a high-quality load-balancing at the second level, we assume that m divides R. A new potential load imbalance exists at the third level of parallelism. If we suppose the cost of one simulation to be a constant, the load will be well balanced only if A divides sp.

A pseudo-random sequence generator is a procedure that starts with a specified random number seed and generates random numbers. We currently use the library PRNGlib [10], which provides several pseudo-random number generators through a common interface on parallel architecture. Common routines are specified to initialize the generators with appropriate seeds on each processor, and to generate in particular uniform distributed random vectors. The proposed generators are successful in most empirical and theoretical tests and have a long period. They can be quickly computed in parallel, and generate the same random sequence independently of the number of processors. This library is used to generate $A \times R$ independent random sequences. It is necessary to make an adequate number of simulation runs so that the mean and standard deviation of the wanted statistics fall within the prescribed error at the specified tolerance. For $R = 32$ and a confidence interval of 95%, the average number of hosts and parasites is known with a relative error of 2%. This is sufficient for a single simulation (without the third level of parallelism) and for the sensitivity analysis on most cases. On the other hand, if a spectral analysis is wanted, $R = 512$ simulation runs are usually performed. The frequency distribution around the mean is then obtained, and constitutes a significant result of the system dynamic.

8 Hybrid OpenMP/MPI Parallelization

Simulations have been performed on an IBM SP3. The machine has 28 NH2 nodes (16-way Power 3, 375 Mhz) with 16 GBytes of memory per node; a Colony switch manages the interconnection of nodes. The code has been developed in FORTRAN 90 with the XL Fortran compiler and using the MPI message-passing library (IBM proprietary version). For performance evaluation and analysis, a representative set of input parameters of a costly simulation were chosen.

First, we evaluate the performances of a single simulation. Let m be the number of MPI processes (parallelization of the r loop in figure 3), nt be the number of OpenMP threads within a MPI process, and $P = m \times nt$ the number of processors ($sp=1$). If $R=32$, the fine-grain parallelism allows us to use more processors than the number of replicates. In our experiments, between one and four OpenMP threads were allocated to compute simulation runs. Figure 4 shows that the execution times decrease for a given number P of processors and an increasing number nt of OpenMP threads (*e.g.* for 32 processors the sequence $m \times nt = 32 \times 1,\ 16 \times 2,\ 8 \times 4$). For these representative results, performances of the MPI-only code always exceed those of the hybrid code. But, we can use 128 processors for $R=32$ with the hybrid code. That means execution times of 81 s on 64 processors and 59,7 s on 128 processors.

Number of threads (nt)	Number of MPI processes (m)				
	1	4	8	16	32
1	3669,9s	935,3 s	471,3s	238,8s	123,8s
2	2385,5s	609,1s	307,5s	155,9s	81,0s
3	1963,5s	500,3s	252,4s	127,8s	67,3s
4	1745,9s	469,4s	228,1s	119,1s	59,7s

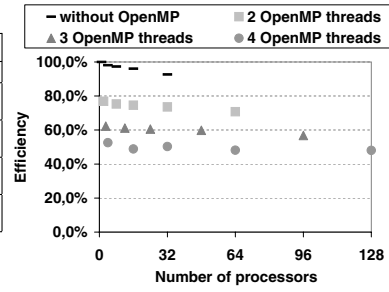

Fig. 4. Execution times and relative efficiency of a simulation for $R=32$; with m MPI process, nt threads in each MPI process, using $m \times nt$ processors

The OpenMP directives add a loop-level parallelism to the simulator. The first level of parallelism consists in the parallelization of a loop (step 2.2.6) with usually many iterations (*e.g.* 2×10^8). The arrays used inside that loop can be shared on the node with hybrid programming. But if we consider an MPI version of this loop-level parallelism, it would imply an overhead due to the communication of these arrays between processors. Precisely, these communication costs would be the main overhead of a MPI implementation.

Furthermore, the computation time spent in that loop represents in average 81 % of the sequential execution time t_{seq}. Let $T_p = 0,81\ t_{seq}$ be the portion of computation time that may be reduced by way of parallelization, and $T_s = 0,19\ t_{seq}$ be the time for the purely sequential part of the program. The Amdhal's

law says, that for n processors the computation time is $T(n) = T_s + \frac{T_p}{n}$. Therefore, the parallel efficiency should be equal theoretically to 84 % for 2 processors (the effective performance is shown on figure 4, $m = 1,\ nt = 2$) and to 64 % for 4 processors ($m = 1,\ nt = 4$). These efficiencies are, in fact, upper limits. They induce a quickly decreasing efficiency for one to several OpenMP threads (nt). A version of our code using POSIX threads was tested and gave the same performances as OpenMP did. In our case, for one parallel loop, there is no overhead between the OpenMP version compared to the POSIX version.

The combination of the first two levels of parallelism were described. In the following, we will focus on the use of the second and third levels, excluding the first one. Each set of processors is not carrying out the same number of simulations. In figure 5, performances of two sensitivity analysis are presented with $A = 15$, $A = 41$.

A=15	Number of processor sets (sp)			
	1	2	4	16
$P = 32$	1677s 100,0%	1725s 97,2%	1754s 95,6%	1697s 98,8%
$P = 64$	–	897s 93,5%	861s 97,4%	861s 97,4%
$P = 128$	–	–	445s 94,2%	438s 95,7%
$P = 256$	–	–	–	223s 94,0%

A=41	Number of processor sets (sp)			
	1	2	4	16
$P = 32$	4444s 100,0%	4607s 96,5%	4531s 98,1%	5438s 81,7%
$P = 64$	–	2319s 95,8%	2298s 96,7%	2566s 86,6%
$P = 128$	–	–	1197 92,8%	1344s 82,6%
$P = 256$	–	–	–	682s 81,4%

Fig. 5. Execution times and relative efficiency of two sensitivity analysis with $A = 15$ and $A = 41$; we use $P = sp \times m$ processors with $R = 32$

The number of processors in one set is at most $R = 32$; we deduce that the maximum number of processors is then $sp \times R$ (impossible configurations are denoted by a minus sign in the tables). For a sensitivity analysis, note that the time is roughly divided by two when the number of processors doubles. For up to 256 processors, really costly simulations can be run with a good parallel efficiency; we can conclude that our implementation is scalable. Nevertheless, efficiency seems lower for $A = 41$ and $sp = 16$. Assume **r**un-**t**imes of **s**imulation runs are close to rts. The sequential complexity comes to $A \times R \times rts$. With the cyclic distribution at the third level, the parallel cost is given by $P \times \lceil A/sp \rceil \times (R \times rts/m)$. This implies an efficiency lower than $A/(sp \times \lceil A/sp \rceil)$. For $A = 41$, $sp = 16$, the parallel efficiency is then theoretically limited up to 85%. The assumption of equal execution times is approximate, but it explains why performances for $A = 41,\ sp = 16$ are not so good. However, an expensive sensitivity analysis ($A = 41$) spends less than 12 minutes on 256 processors.

9 Biological Results

The results given by the new stochastic and the deterministic simulators come from two distinct computation methods. Both are based on a single bio-mathe-

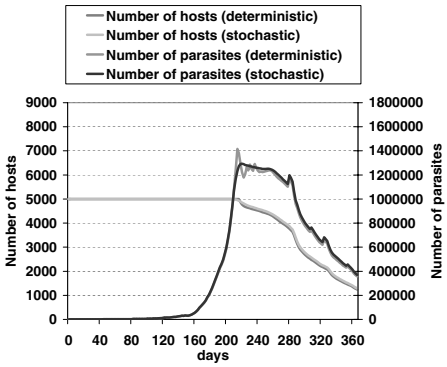

Fig. 6. Experiment with temporary endemic state at the end of simulation

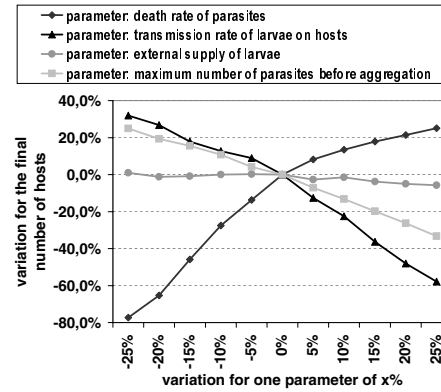

Fig. 7. Sensitivity analysis for 41 sets of input parameters

matical model and then outputs should not be very different. In fact, similarities are clearly observed, the number of hosts and parasites are given for one experiment in figure 6. For the stochastic simulation, the mean of $R=400$ simulation runs is given. For some parameter values, variations are observed between the two simulators. We already know that some interactions in the host-parasite system cannot be reproduced in the deterministic simulator (without modeling at a finer scale). Figure 7 corresponds to one result of the sensitivity analysis introduced in figure 5 ($A=41$); the intersection point (0%,0%) corresponds to a reference simulation. It shows the variation in percentage of the final number of hosts depending on the variation of four distinct input parameters. We conclude that the system is very sensitive to the death rate of parasites.

10 Conclusion

For similar outputs, a complete and costly stochastic simulation of the host-parasite system lasts only 1 minute on 128 processors versus 28 minutes for the deterministic simulation. A performance analysis has established the efficiency and the scalability of the stochastic algorithm using three levels of parallelism. The hybrid implementation allows us to use more processors than the number of simulation runs. The stochastic simulation gives the frequency distribution around the mean for outputs, providing new insights into the system dynamics. The sensitivity analysis, requiring several series of simulations is now accessible. An expensive sensitivity analysis spends less than 12 minutes on 256 processors.

References

1. M. Bernaschi, F. Castiglione, and S. Succi. A parallel algorithm for the simulation of the immune response. In *WAE'97 Proceedings: Workshop on Algorithm Engineering, Venice Italy*, September 1997.

2. M.W. Berry and K.S. Minser. Distributed Land-Cover Change Simulation Using PVM and MPI. In *Proc. of the Land Use Modeling Workshop, 1997*, June 1997.
3. C. Bouloux, M. Langlais, and P. Silan. A marine host-parasite model with direct biological cycle and age structure. *Ecological Modelling*, 107:73–86, 1998.
4. M. Flechsig. Strictly parallelized regional integrated numeric tool for simulation. Technical report, Postdam Institue for Climate Impact Reasearch, Telegrafenberg, D-14473 Postdam, 1999.
5. M. Langlais, G. Latu, J. Roman, and P. Silan. Parallel numerical simulation of a marine host-parasite system. In P. Amestoy, P. Berger, M. Daydé, I. Duff, V. Frayssé, L. Giraud, and D. Ruiz, editors, *Europar'99 Parallel Processing*, pages 677–685. LNCS 1685 - Springer Verlag, 1999.
6. M. Langlais and P. Silan. Theoretical and mathematical approach of some regulation mechanisms in a marine host-parasite system. *Journal of Biological Systems*, 3(2):559–568, 1995.
7. G. Latu. Solution parallèle pour un problème de dynamique de population. *Technique et Science Informatiques*, 19:767–790, June 2000.
8. H. Lorek and M. Sonnenschein. Using parallel computers to simulate individual-oriented models in ecology: a case study. In *Proceedings: ESM'95 European Simulation Multiconference, Prag*, June 1995.
9. B. Maniatty, B. Szymanski, and T. Caraco. High-performance computing tools for modeling evolution in epidemics. In *Proc. of the 32nd Hawaii International Conference on System Sciences*, 1999.
10. N. Masuda and F. Zimmermann. *PRNGlib : A Parallel Random Number Generator Library*, 1996. TR-96-08, ftp://ftp.cscs.ch/pub/CSCS/libraries/PRNGlib/.
11. OpenMP. A Proposed Industry Standard API for Shared Memory Programming. October 1997, OpenMP Forum, http://www.openmp.org/.
12. P. Silan, M. Langlais, and C. Bouloux. Dynamique des populations et modélisation : Application aux systèmes hôtes-macroparasites et à l'épidémiologie en environnement marin. In C.N.R.S eds, editor, *Tendances nouvelles en modélisation pour l'environnement*. Elsevier, 1997.