Benefits of parallel processing in stochastic simulations of fish breeding programs

Ingrid Olesen[1] and Jørn Amundsen[2]
[1]AKVAFORSK, P.O. Box 5010, N-1432 Ås, Norway
[2]Norwegian University of Science and Technology, N-7034 Trondheim, Norway
e-mail: ingrid.olesen@akvaforsk.nlh.no

Abstract

Experiences with adapting and running a Fortran 77 program for stochastic simulation of a mass selection program in fish breeding for parallel processing are presented.

Little rewriting of the program was required to assign different replicates of the simulations to different processors. In order to study the speedup according to number of processors, a case of 20 replicates, requiring 2 000 s using one processor, was run. Wall clock time for the program (without initial input) using 1, 2, 4, 10 and 20 processors were obtained.

The saved computer time increased considerably with increased number of processors. Wall clock time was 17 times lower on a Cray T3E using 20 parallel processes compared to sequential execution (116 s versus 2000 s). This is expected to be improved for a more realistic and demanding case.

1. Introduction

Stochastic simulations are very useful for optimising selection breeding programs with respect to genetic gain and inbreeding. They are however demanding with respect to computing time, especially when inbreeding coefficients of all individuals and many replicates are required. Super computers with multiple processors may therefore be very efficient for running such programs.

The objective of this paper was to present our experiences with adapting and running a Fortran 77 program for stochastic simulation of mass selection on a Cray T3E with parallel processing.

2. Material and methods

Simulation programme fisksim

A Fortran 77 computer programme was developed for simulating mass selection in a fish breeding programme over a number of generations. Stochastic simulations were applied to study genetic changes and rates of inbreeding for various family structures, various numbers of families up to 100, family sizes up to 150 and various sets of genetic parameters. The output file with records of breeding values and inbreeding for all individuals made the basis for these studies. Although a fast subroutine was applied for calculation of inbreeding coeffisients, this made up most of the computing time.

Parallelisation of Fisksim

The fish breeding program, «Fisksim» was parallelised employing functional decomposition or work sharing. This strategy shares work between processors, each processor working on independent data.

Work sharing is often simpler to implement than data or domain decomposition, but tend to duplicate data among processors and hence increase the total need for memory.

On physically distributed memory computers, as used here, available node memory is often a limited resource. The work sharing approach may then limit the usefulness of employing a large number of processors to solve larger problems since each piece of work may grow out of local node memory. However, Fisksim has moderate per node memory requirements, so this is not an issue for the application studied here.

Co-Array Fortra (CAF) [1,2] is chosen to parallelise the program. This could easily be done with other approaches like MPI (Message-Passing Interface) [3], but CAF was chosen because it is available on the target system and caused a minimum of distortions to the sequential program.

Each invocation of Fisksim runs a number of replicates (NREPL), where each replicate contains a number of generations (NAAR):

```
DO IREP = 1, NREP
    repseed = update_repseed()

    cycleseed = repseed
    init_cycle()

    DO IAAR = 1, NAAR
       perform_cycle(IAAR)
    END DO ! IAAR
END DO ! IREP
```

Fisksim employs two different random number generators (RNG's). The first RNG generates the starting seed of the second RNG employed to cycle through the specified number of generations of fish breeding (NAAR). When running in parallel, the update_repseed is replicated on every processor so that the cycle seed sequence is identical with that of a sequential execution. Except for the initial input each replicate cycle (IREP) is an independent piece of work. Having read the simulation parameters and initial seed from the input file, the parallelisation strategy is to replicate the update_repseed() across every processor. Work sharing of replicates across processors is done as follows:

```
MYIMG = THIS_IMAGE()
   NIMG = NUM_IMAGES()
   ...
   DO IREP=MYIMG,NREPL,NIMG
...
END DO ! IREP
```

In the pseudo code above, NIMG is the number of processors, or images, and MYIMG = 1...NIMG designates an individual image.

Of course this approach will have poor load balancing if the number of replicates is not divisible by the number of processors employed. This is a minor limitation, however. Due to the simple strategy of parallelisation with minimal interprocessor communication, Fisksim should also perform very well on a Beowulf system (cluster of PC's).

Test case

In order to study the speedup according to number of processors, a case of 50 families and 30 progeny per family bred for 15 generations was run. For running 20 replicates using one processor, it required 2 000 seconds. Wall clock time for the program without initial input using 1, 2, 4, 10, and 20 processors were obtained. Speedup for N processors was calculated as the wall clock time using one processor (Time1) divided by the wall clock time using N processors (TimeN): Speedup = Time1/TimeN.

Regression analysis

In order to test the linearity of the speedup, a simple linear regression analysis was carried out. The regression of speedup on number of processors was analysed, and regression coeffisient, standard error, predicted speedup and standard error of predicted values were obtained.

3. Results and discussion

As shown in Table 1 and Figure 1, the computing time was considerably reduced with increased number of processors. Wall clock time was 17 times lower on a Cray T3E using 20 parallel processes compared to sequential execution.

Table 1. Wall clock time for simulating 20 replicates and speedup according to number of processors

No. processors	Wall clock time, s	Speedup
1	2 000	1.00
2	1 039	1.92
4	487	4.11
10	208	9.64
20	116	17.29

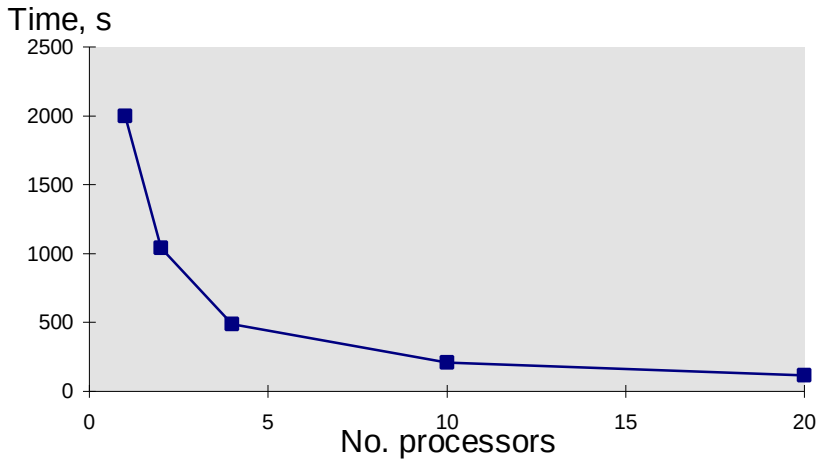

Figure 1. Wall clock time for simulating 20 replicates according to number of processors.

Obtained speedup is shown in Table 1 and depicted in Figure 2. As can be seen it deviates little from a linear speedup with slope equal to 1. The deviation is however larger for 20 processors. At 20 processors, queuing on output resources start to become significant compared to the computing time. Nothing has been done to optimise output for a large number of processors. This could be done, but will only be of little interest in real cases compared to our small scale benchmarking case. For this small case, the simultaneous output from 20 processors therefore made a larger proportion of the computing time than from fewer processors. Hence, the performance is expected to be improved for a more realistic and demanding case, as the output will make up a smaller proportion of the job.

The regression of speedup on processor number was estimated to 0.86 ± 0.03. Due to the observation of 20 processors, it was a little lower than unity. Predicted speedup with 99 % confidence interval is shown in Figure 3. As can be seen the linear speedup with slope of unity is also within the confidence interval.

4. Conclusions

Little rewriting of the program was required to assign different replicates of the simulations to different processors. The computer time decreased considerably with increased number of processors with an almost linear speedup. The wall clock time was approximately 17 times lower on a Cray T3E using 20 parallel processes compared to sequential execution. The slope of the speedup obtained was close to unity.

Acknowledgement

This work has received support from The Research Council of Norway (Programme for Supercomputing) through a grant of computing time.

References

1. Numrich,R.W. and J. Reid, 1998. Co-Array Fortran for Parallel Programming, ACM Fortran Forum, Vol 17: no 2, pp 1-31.
2. Co-Array Fortran. http://www.co-array.org/
3. MPI: A Message-Passing Interface Standard, Message Passing Interface Forum, May 5, 1994. http://www.mpi-forum.org/

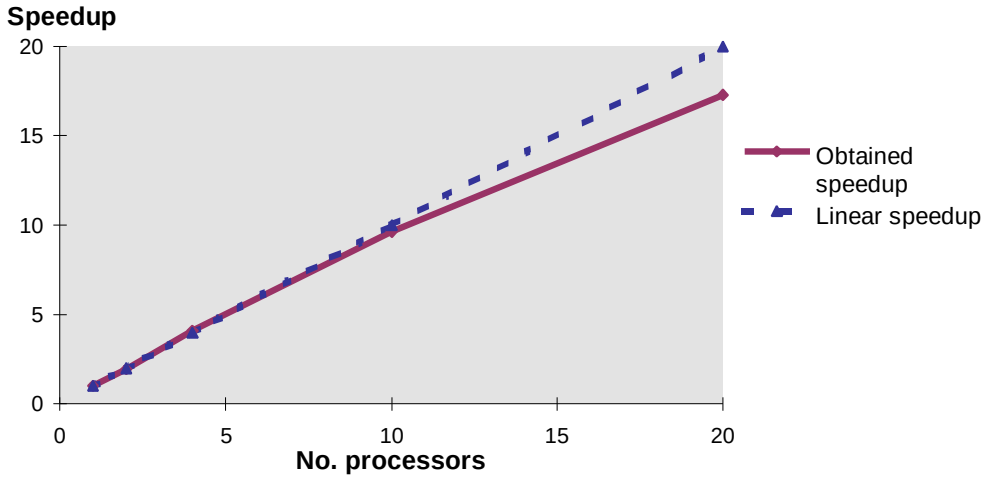

Figure 2. Obtained speedup and linear speedup with unity slope according to no. processors.

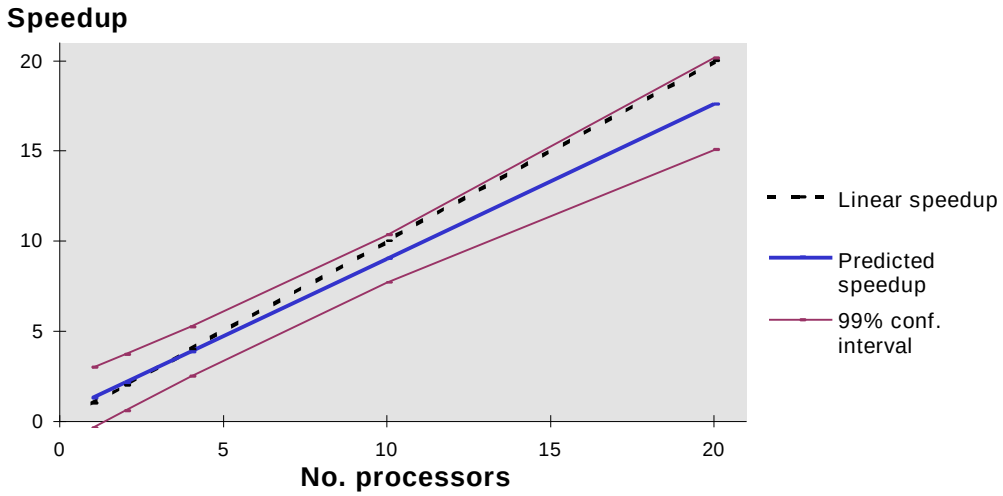

Figure 3. Predicted speedup according to no. processors with 99 % confidence interval. Linear speedup with slope equal to 1 is also indicated.