# ADAPTING AN EVOLUTIONARY ALGORITHM WITH EMBEDDED SIMULATION AND PSEUDO-RANDOM NUMBER GENERATION FOR THE CELL BROADBAND ENGINE

Sofie Van Volsem, Sven Neirynck
Department of Industrial Management
Ghent University
Technologiepark 903
BE-9052 Zwijnaarde, Belgium
e-mail: {Sofie.VanVolsem, Sven.Neirynck}@UGent.be

## KEYWORDS

PlayStation®3, cell processor, SIMD, normal random numbers, simulation, evolutionary algorithm

## ABSTRACT

For the problem of optimizing inspection strategies in multi-stage production systems, a metaheuristic consisting of an evolutionary algorithm with embedded simulation was developed in Van Volsem et al. (2007), Van Volsem (2009) and Van Volsem (accepted for publication, 2009). The metaheuristic requires normally distributed pseudo-random numbers; the time needed for this random number generation is a substantial fraction of the total computation time. In an effort to reduce the computation time, the metaheuristic was adapted for computation on the Cell Broadband Engine. The proposed adaptation is twofold: we propose a way to make the metaheuristic suitable for fast multicore computation, and secondly, the potential of SIMD computation for speeding up the random number generation process and the metaheuristic is investigated.

## INTRODUCTION

Traditional computer software is written for serial computation. To solve an optimization problem, an algorithm or metaheuristic is constructed and implemented as a serial stream of instructions. These instructions are executed on a central processing unit (CPU) on one computer.

Parallel computing uses multiple processing elements simultaneously to solve a problem. This is accomplished by breaking the problem into independent parts so that each processing element can execute its part of the algorithm simultaneously with the others. The processing elements can be diverse and include resources such as a single computer with multiple processors, several networked computers, specialized hardware, or any combination of the above.

Today most commodity CPU designs include single instructions for some vector processing on multiple (vectorized) data sets, typically known as SIMD (Single Instruction, Multiple Data). Modern video game consoles and consumer computer-graphics hardware rely heavily on vector processing in their architecture. In 2000, IBM, Toshiba and Sony collaborated to create the Cell Broadband Engine (Cell BE), consisting of one traditional microprocessor (called the Power Processing Element or PPE) and eight SIMD co-processing units, or the so-called Synergistic Processor Elements (SPEs), which found use in the Sony PlayStation®3 among other applications.

The computational power of the Cell BE or PlayStation®3 can also be used for scientific computing. Examples and applications have been reported in e.g. Kurzak et al. (2008), Bader et al. (2008), Olivier et al. (2007), Petrini et al. (2007).

In this paper, the potential of using the PlayStation®3 for speeding up metaheuristic optimization is investigated. More specifically, we propose an adaptation of an evolutionary algorithm with embedded simulation for inspection optimization. Thereto, two issues need to be addressed:

- the metaheuristic itself needs to be adapted to make it suitable for parallel computation and vector processing, and
- the random number generation required by the metaheuristic has to be adapted for parallel computing as well.

The main mechanism of achieving this goal is through parallelization and vectorization. The main obstacles in the way of parallel execution are data hazards, which prevent simultaneous execution of instructions with dependencies between their arguments, and control hazards, which result from branches and other instructions changing the Program Counter (Kurzak et al. 2008). In other words:

- Instructions can be grouped together only if there is no data dependency between them.

- Communication and synchronization between the different subtasks are typically one of the greatest obstacles to getting good parallel program performance.

Moreover, the potential speed-up of an algorithm on a parallel computing platform is limited by Amdahl's law, which states that a small portion of the program which cannot be parallelized will limit the overall speed-up available from parallelization.
The remainder of the paper is organised as follows: in Section 2, the original problem and metaheuristic solution approach is described. Section 3 reports on the adaptation of the metaheuristic, while the potential of SIMD computation for speeding up the random number generation process is investigated in Section 4. Section 5 summarizes and concludes.

## INSPECTION OPTIMIZATION WITH AN EVOLUTIONARY ALGORITHM AND SIMULATION

Efficient production quality control is a major issue to manufacturers. Efficient economic inspection strategies ensure the required output quality while minimizing the total inspection cost. Generally speaking, more and tighter inspection will induce a higher product quality – in terms of meeting product specifications– but will also result in higher costs of inspection, scrap and rework. An economic inspection plan will balance this trade-off.
Consider a serial multistage production system (MSPS) in which products travel sequentially from stage 1 to stage $n$ and inspection of products is performed by $k$ ($k \leq n$) inspection stations. At each stage, a manufacturing action is performed on or with the products, before moving on to an inspection station, or to the processing station of the next stage in case of no inspection.
An inspection strategy for an MSPS decides on:

1. the number and location of inspection stations;

2. the rigor of the inspections (inspection limits) for each inspection station.

3. the number of inspections executed (sample size or sampling frequency and acceptance number) for each inspection station;

The problem facing the MSPS inspection planner thus consists of finding the combination of these inspection parameters in order to minimize the total expected inspection cost ($TIC$). This is a complex joint optimization problem; addressed in Van Volsem et al. (2007), Van Volsem (2009) and Van Volsem (accepted for publication, 2009).
Their joint optimization method consists of embedding Monte Carlo simulation (to compute the serial $n$-stage MSPS subject to inspection) in an Evolutionary Algorithm (EA) (to perform the actual optimization). For each of the $n$ stages, the EA proposes the inspection type ($F$ for full inspection, $S$ for sampling inspection, $N$ for no inspection), the upper and lower inspection limits ($UIL$ and $LIL$), and the acceptance sampling parameters ($s$ and $t$) where appropriate.
The following notation is used:

| | | |
|---|---|---|
| $K$ | = | batchsize |
| $n$ | = | number of process stages |
| $p'_i$ | = | fault occurrence in stage $i$ |
| $LIL_i$ | = | lower inspection limit in stage $i$ (variable) |
| $UIL_i$ | = | upper inspection limit in stage $i$ (variable) |
| $LS_n$ | = | lower specification limit after stage $n$ (fixed) |
| $US_n$ | = | upper specification limit after stage $n$ (fixed) |
| $s_i$ | = | sample size for stage $i$ |
| $t_i$ | = | acceptance number for stage $i$ |
| $d_i$ | = | number of bad items after stage $i$ |
| $c_{T,i}$ | = | unit test cost in stage $i$ |
| $c_{R,i}$ | = | unit rework cost in stage $i$ |
| $c_P$ | = | unit penalty cost (after stage $n$) |
| $TC_i$ | = | test cost in stage $i$ |
| $RC_i$ | = | rework cost in stage $i$ |
| $\alpha_{F,i}$ | = | 1 if $F$ is selected in stage $i$, = 0 otherwise |
| $\alpha_{S,i}$ | = | 1 if $S$ is selected in stage $i$, = 0 otherwise |
| $TTC$ | = | total test cost |
| $TRC$ | = | total rework cost |
| $TPC$ | = | total penalty cost |
| $TIC$ | = | total inspection cost |

The $TIC$ is calculated as follows:

$$TIC = TTC + TRC + TPC \quad (1)$$

with

$$TTC = \sum_{i=1}^{n} TC_i \quad (2)$$

$$TRC = \sum_{i=1}^{n} RC_i \quad (3)$$

$$TPC = c_P . d_n \quad (4)$$

and with

$$TC_i = c_{T,i} . (\alpha_{F,i} . K + \alpha_{S,i} . s_i) \quad (5)$$

$$RC_i = c_{R,i} . p'_i . \alpha_{F,i} . K \quad (6)$$

Determining the optimal inspection strategy, i.e. the whole of inspection decisions that minimize the $TIC$, requires the determination of inspection options $\alpha_i$ and the corresponding inspection limits ($LIL_i$, $UIL_i$) and sampling parameters ($s_i$, $t_i$), for all stages $i = 1, ..., n$.
This is what the proposed Evolutionary Algorithm does.
Evolutionary Algorithms are adaptive heuristic search methods mimicking selective breeding, where offspring

are sought which have certain desirable characteristics, determined at the genetic level by combination of the parents' chromosomes. In a similar way, in seeking better solutions, EA's combine pieces of existing solutions: new generations of offspring are generated through an iteration process until a convergence criterion is met. The basic concepts were developed by Holland (1975) and were forged into a problem solving methodology for complex optimization problems by De Jong (1975) and Goldberg (1989).

There are four main parts in the EA paradigm, namely the problem representation and initiation, the objective function evaluation (fitness calculation), the parent selection, and the actual evolutionary reproduction of candidate solutions.

### Problem representation and initiation

Every proposed solution is represented by a vector of the independent variables (inspection decision variables), coded as a chromosome constituted by as many *genes* as the number of independent variables. The chromosomes used in the EA we propose, consist of a set of "character" values ($F$, $N$ or $S$), real values ($LIL_i$ and $UIL_i$) and integer values ($s_i, t_i$).

We used a population size $M$ of 50 initial solutions. From this pool, some are selected (parents) to construct new solutions (children). The construction algorithm for the initial population consists in randomizing the characters ($N, S, F$), and randomizing the limits by allowing (symmetrical) variation from the original limits by a certain user defined percentage.

### Objective function evaluation (fitness calculation)

For every candidate solution its fitness as a possible parent has to be evaluated, where fitness refers to measure of profit or goodness to be maximized while exploring the solution space. We use a straightforward normalization procedure to calculate the fitness value $f$ for each solution $j$ in a population of $M$ solutions:

$$f_j = \frac{1/\,TIC_j}{\sum_{k=1}^{M}(1/TIC_k)} \tag{7}$$

### Parent selection

Parent selection for producing offspring is done as in Holland's original Genetic Algorithm, i.e. for each reproduction two parents are chosen: one parent is selected on its fitness basis, the other is chosen randomly. The idea behind this is that the parent chosen for its fitness ensures genetic quality, while the random parent ensures genetic diversity.

### Reproduction

In our algorithm, the new generation consists of $(M-1)$ children, the $M^{th}$ solution in the next generation population is the best solution from the previous generation (=*elitism of 1*). Generating offspring is performed in two steps: first crossover is applied, then the inspection limits are adapted. After these two steps, reproduction is completed and the children thus obtained can populate the new generation. This way, the simultaneous determination of inspection parameters is achieved.

Our crossover operator randomly selects a crossover point, and constructs two new solutions by exchanging the tails of both parents. Instead of mutation, inversion is used (see Reeves (1993; pg. 173)).

**Algorithm 1** original EA

```
Create initial sorted population
for generation = 1 to number of generations do
  Create offspring
  for solution = 1 to $M$ do
    for stage = 1 to $n$ do
      Calculate process values
      Calculate inspection cost
    end for
    Calculate TIC
  end for
  Sort population
end for
Take winner
```

## ADAPTATION OF THE EA

Kurzak et al. (2008) state that while not all problems SIMDize well, most can benefit from it one way or another.

To adapt the above metaheuristic for computation on the Cell Broadband Engine, we need to parallelize it to make it suitable for computation on the SPEs, and consequently the code has to be vectorized for in order to make efficient use of the SPEs.

First we have to choose which parts will run on the SPEs and what on the PPE. Next step is to SIMDize the SPE code. Caclulating the TICS of an inspection strategy requires a fair amount of computational power and has no data dependencies. This is therefore an ideal candidate to run on the SPE's. The creation of offspring for new generations will be done on the central PPE, which will then communicate the inspection strategies to the SPEs who will calculate the TICs in parallel.

The TIC calculation on the SPE's consists of calculating the process values and subject them to the selected inspection strategy. Each solution is simulated 50 times, the average TIC is returned. This simulation requires a lot of normal random numbers, how to adapt the random number generation for parallel processing is the

**Algorithm 2** adapted version of the EA

```
Create initial sorted population
for generation = 1 to number of generations do
  for solution = 1 to $M$ do
    Create offspring
    Calculate TIC on SPE (IN PARALLEL)
  end for
  Sort population
end for
Take winner
```

subject of the next section.
How is the TIC calculation now implemented on an SPE? Keeping in mind that the memory of an SPE is limited to 256k, we have decided to calculate it one process value at a time, through all $n$ stages instead of complete batches stage by stage. This is feasible as the processing of the values is not interdependent. This way we needn't worry about the 256k memory which would otherwise become a problem if the batchsize becomes to big.
How do we SIMDize the inspection? Normally this code consists of a lot of branches:

```
Do we need to inspect? YES/NO
Is the value between inspection limits? Y/N
Are we using sampling inspection? Y/N
Do we need to switch to full inspection? Y/N
```

In SIMD we cannot have the conditional branches as the same code needs to run on all the elements in the vector. We implemented a standard technique of processing the instructions for both branches of the conditional branch and at the end select the right value with the instruction `SPUsel`.

## NORMAL PSEUDO-RANDOM NUMBERS

In many simulation and Monte Carlo programs, a substantial fraction of the computation time is used in generating pseudo-random numbers (Brent 1998). Vector or parallel computation can significantly contribute in accelerating the simulation process. However, parallel computation for Monte Carlo programs in itself also brings about some difficulties that cannot be overlooked:

- The requirements for parallel random number generators (RNGs) are more stringent than those for sequential RNGs. If a simulation is to be run on a multi-processor machine, it is of the essence to ensure that the random numbers used by each processor are independent, or equivalently, to ensure that the sequences of random numbers used by each processor are disjoint.

Different applications require pseudo-random numbers with different distributions (uniform, normal, exponential, Poisson, etc.). The algorithms used to generate these random numbers usually rely on a good source of uniform random numbers, which are then transformed to random numbers with other distributions. The algorithms used can be divided in the following groups:

**Inversion methods** are based on the observation that continuous cumulative distribution functions (cdfs) range uniformly over the interval $[0, 1]$. If $u$ is a uniform random number on $[0, 1]$, then a random number $X$ from a continuous distribution with specified cdf $F$ is obtained using $X = F^{-1}(U)$. Subject to the restriction that the distribution is continuous, this method is generally applicable (and can be computationally efficient if the cdf can be analytically inverted)

**Transformation methods** provide an alternative in cases where cdf inversion too computationally expensive in practice for some probability distributions. The Box-Müller transform is an example of such an algorithm. It produces two normally distributed random numbers from a pair of uniformly distributed random numbers. If $u1$ and $u2$ are independent random variables that are uniformly distributed on [0, 1], then

$$\begin{aligned} z_0 &= R\cos(\Theta) = \sqrt{-2\ln u_1}\cos(2\pi u_2) \\ z_1 &= R\sin(\Theta) = \sqrt{-2\ln u_1}\sin(2\pi u_2) \end{aligned}$$

are independent random variables with a standard normal distribution.

**Acceptance-rejection methods** also provide an alternative in cases where the functional form of the required distributions makes it difficult or time-consuming to generate random numbers using inversion methods. As the previous two methods, acceptance-rejection methods require uniform random numbers. In this method it is assumed that the probability distribution $F$ we wish to simulate has a pdf $f(x)$. The basic idea is to find an alternative probability distribution $G$, with pdf $g(x)$, from which we already have an efficient algorithm for generating from, but also such that the function $g(x)$ is "close" to $f(x)$. In particular, we assume that the ratio $f(x)/g(x)$ is bounded by a constant $c > 0$. The algorithm for generating $X$ distributed as $F$ proceeds as follows:

1. Choose a pdf $g$.
2. Find a constant $c$ such that $f(x)/g(x) \leq c \,; \forall\, x$
3. Generate a uniform random number $u$
4. Generate a random number $v$ from $g$
5. If $c * u \leq f(v)/g(v)$ , accept and return $v$. Otherwise, reject $v$ and go to step 3

For efficiency, a "cheap" method is required for generating random numbers from $g$, and the scalar $c$ should be small.

These methods all either involve the computation of mathematical functions such as sines, cosines and logarithms, which are slow in comparison to the time required to generate a uniform random number, or require on average more than one uniform random number for each normal random number. From this it evidently follows that normal RNGs based on transforming uniform random numbers are slower than uniform RNGs. Leva (1992) compared several of the best acceptance-rejection methods an found that they are at least five times slower than a fast uniform RNG on the same machine.

Brent (1998) argues that the most well-known and widely used methods for normal RNG often do not vectorize well. He therefore suggests vectorized implementations of the "old-fashioned" Box-Müller transformation.

We follow this suggestion; the RNG used to generate the uniform random numbers is the SFMT (SIMD-oriented Fast Mersenne Twister, Saito and Matsumoto (2006)), followed by our own implementation of the Box-Müller transform. Standard mathematical functions such as sines and cosines are calculated using the Universal SIMD Mathlibrary (2009) `libsimdmath`.

## SUMMARY AND CONCLUSIONS

We design an optimized parallel implementation of an evolutionary algorithm and simulation for optimizing inspection strategies for multi-stage processes on the PlayStation®3. We adapted the algorithm to eliminate branches and optimized the code using standard techniques such as loop unrolling and vectorization. We adapted the random number generation process: we developed and implemented a Box-Müller transform on uniform random numbers generated with an 128-bit Mersenne twister. The original algorithm calculation time was >1 hour for 200 generations; re-writing the code with SIMDizing the RNG led to a calculation time of 5'59" on a single core of a 2.5GHz AMD Phenom processor. The further porting and optimizing of the code to make it suitable for running on the cell broadband engine led to a calculation time of 14" on a PlayStation®3. We thus realized a speedup factor of more then 256 in comparison with the original algorithm in Van Volsem et al. (2007), and showed the PlayStation®3 suitable for scientific computing.

## REFERENCES

Bader D.; Chandramowlishwaran A.; and Agarwal V., 2008. *On the design of fast pseudo-random number generators for the cell broadband engine and an application to risk analysis. IEEE Transactions on the 37th International Conference on Parallel Processing.*

Brent R., 1998. *Random number generation and simulations on vector and parallel computers. Proceedings of the 4th International Euro-Par Conference*, 1–20.

De Jong K.A., 1975. *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems.* Ph.D. thesis, University of Michigan Press.

Goldberg D., 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley, NY.

Holland J.H., 1975. *Adaptation in Natural and Artificial Systems.* University of Michigan Press.

Kurzak J.; Buttari A.; Luszczek P.; and Dongarra J., 2008. *The PlayStation3 for high performance scientific computing. Computing in Science and Engineering*, 10, no. 3, 84–87.

Leva J., 1992. *A fast normal random number generator. ACM Transactions on Mathematical Software*, 18, 449–453.

Olivier S.; Prins J.; Derby J.; and Vu K., 2007. *Porting the GROMACS Molecular Dynamics Code to the Cell Processor. Proceedings of the 8th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing.*

Petrini F.; Fossum G.; Fernandez J.; Varbanescu A.L.; Kistler M.; and Perrone M., 2007. *Multicore surprises: lessons learned from optimizing Sweep3D on the cell broadband engine. IEEE Transactions on the 2007 International Parallel and Distributed Processing Symposium.*

Reeves C.R., 1993. *Modern Heuristic Techniques for Combinatorial Problems.* Blackwell Scientific Publications.

Saito M. and Matsumoto M., 2006. *SIMD-oriented Fast Mersenne Twister: a 128-bit Pseudorandom Number Generator. Proceedings of the the 7th International Conference on Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing.*

Universal SIMD Mathlibrary, 2009. URL `http://webuser.fh-furtwangen.de/~dersch/`.

Van Volsem S., 2009. *Joint optimization of all inspection parameters for multi-stage processes: algorithm, simulation and test set. Proceedings of the 16th European Concurrent Engineering Conference.*

Van Volsem S., accepted for publication, 2009. *Joint optimization of all inspection parameters for multi-stage processes: evolutionary algorithm and simulation. International Journal of Innovative Computing and Applications.*

Van Volsem S.; Dullaert W.; and Van Landeghem H., 2007. *An Evolutionary Algorithm and Discrete Event Simulation for Optimizing Inspection Strategies for Multi-Stage Processes. European Journal of Operational Research*, 179, 621–633.

## AUTHOR BIOGRAPHY

**SOFIE VAN VOLSEM** received a MSc degree in Chemical Engineering from Ghent University in 1998 and a PhD in Engineering Sciences from the same institution in 2006. She worked in industry as a process & quality engineer before returning to academia. After being with the University of Antwerp for 6 years and the University College of West-Flanders for nearly 2 years, she currently holds a post-doc position at Ghent University. She teaches Quality & Industrial Statistics. Her research interests are quality management, quality and reliability issues in supply chains, management applications of metaheuristics.

**SVEN NEIRYNCK** graduated as MSc in Computer Sciences at Ghent University in 1997. After 10 years of working as a systems engineering manager, he currently divides his time between IT consultancy with expertise in storage, archiving, data security and HPC systems; and research in the area of simulation at Ghent University.