Implementation and Optimization of cGA+LS to solve Capacitated VRP over Cell/B.E.

Asim Munawar [*1], Mohamed Wahib [*1], Masaharu Munetomo [*2], Kiyoshi Akama [*2]
[*1, Corresponding author] *Graduate School of Information Science and Technology Hokkaido University, Sapporo 060-0811, JAPAN*
[*2,] *Information Initiative Center, Hokkaido University, Sapporo 060-0811, JAPAN*
{asim,wahibium}@uva.cims.hokudai.ac.jp, {munetomo,akama}@iic.hokudai.ac.jp
doi: 10.4156/ijact.vol1.issue2.2

Abstract

This paper presents a case study to illustrate the design and implementation of cellular Genetic Algorithm (cGA) with Local Search (LS) to solve Capacitated Vehicle Routing Problem (CVRP) over Cell Broadband Engine (Cell BE). Cell BE is a heterogeneous, distributed memory multicore processor architecture for multimedia applications with regular memory access requirements. It has one 64-bit Power Processing Element (PPE) that acts as the main processor and 8 Synergistic Processing Elements (SPEs) with only 256 KB of local memory, each for instructions and data. GAs on the other hand use population based search techniques. Such techniques usually have large memory requirements and show non-uniform memory access patterns. These properties of GAs make their implementation over Cell BE even more challenging. In order to take maximum advantage of the hardware, we propose an asynchronous approach to implement cGA+LS over Cell BE. In this paper, we discuss the implementation and optimization of the proposed method in detail. We compare the proposed method with other state-of-the-art CVRP solvers and synchronous implementation of cGA+LS over Cell BE. We solve existing benchmark problems and achieve considerable speedups. We extend the work further to solve extremely large instances of CVRP compared to ones present in the CVRP literature, and get acceptable results in a reasonable amount of time.

Keywords

Cellular Genetic Algorithm (cGA); Combinatorial Optimization Problem; Multicore; Cell Broadband Engine Architecture; Vehicle Routing Problem.

1. Introduction

The Vehicle Routing Problem (VRP) is a combinatorial optimization problem, seeking to service a certain number of customers with a fleet of vehicles. Real-world applications have widely shown that the use of computerized procedures for the distribution process planning, generally results in 5% to 20% savings in the global transportation cost [1]. As transportation cost represents 10% to 20% of the final cost of the goods [1], therefore, VRP is a widely studied problem both theoretically and in practice [2]. There are different variants of VRP which pose different constraints on delivery time and route length. In this paper, we consider a well-known variant of VRP known as Capacitated Vehicle Routing Problem (CVRP). In CVRP each customer has a fixed demand of goods and each vehicle has a limited carrying capacity, moreover, the pickup point is always a central depot. CVRP can be of two types: CVRP with time windows and CVRP without time windows. In this paper, we only consider CVRP without time windows, i.e. there is no constraint on the time of the delivery (we will use the term CVRP to refer CVRP without time windows throughout this paper unless specified otherwise).

Exact methods of CVRP can be used to solve a CVRP with less than 50 customers [1]. However, for problems involving more than 50 customers, approximate methods e.g. heuristics and metaheuristics, have gained more attention over the last few years. Algorithms like Tabu Search [3], Simulated Annealing [4], and Genetic Algorithms (GA) [5] gained a lot of attention during this time. GAs are comparatively simple and highly customizable as compared to their other counterparts, making them one of the most important type of optimization algorithms. In this paper, we have used cellular GA (cGA) with a structured population as the base of the algorithm. In order to improve the results, we have used two well-known Local Search (LS) techniques, namely 2-Opt [6] and λ-Interchange [7]. It is important to note that

LS is mandatory to get good results for CVRP using GAs [2,8,9,10,11]. LS is also mandatory for a wide variety of problems to improve convergence. This hybrid of cGA+LS to solve CVRP, was first suggested by E. Alba et al. [2], and it has the potential to compete with any state-of-the-art CVRP algorithm available.

GAs are algorithms inspired from the natural process of evolution. They maintain a population of probable solutions and apply genetic operators in search for better solutions over the generations. GAs give above average performance for a wide range of optimization problems, at the expense of computational power they use. Therefore, it is common to implement GAs over High Performance Computing (HPC) clusters. An HPC cluster provides enhanced performance by splitting the computational task (GAs in this case) among the nodes in the cluster. Clusters, although being cost effective, are greatly influenced by hardware constraints, leading to the three performance-limiting walls [12] namely power wall, memory wall, and instruction level parallelism wall. These three performance limiting walls combine to limit performance growth for single-core microprocessors and hence, for clusters consisting of single-core processing nodes.

A multi-core architecture is one of the solutions to tackle the three wall problem. This has led most microprocessor vendors to turn instead to multicore chip organizations, even though the benefits of multiple cores can only be realized if the programmer or compiler explicitly parallelize the software [13]. Multicore processors are broadly divided into two categories: 1) homogeneous, and 2) heterogeneous. Homogeneous multicore processors usually have identical processing cores and share a common memory space, on the other hand, heterogeneous processors have heterogeneous cores and the memory is usually distributed among the available cores. Homogeneous multicore processors are considered to be general purpose and are usually easier to program. They have large caches and complex control logic. Heterogeneous processors are mostly designed for specific applications. They usually lack big caches and complex control logic; instead they use more transistors to provide computational modules. Therefore, commodity heterogeneous processors have 10-100's of processing cores as compared to less than 10 cores in a comparable grade homogeneous multicore processor. As a consequence, although difficult to program, heterogeneous processors can achieve better speedups as compared to a similar grade homogeneous processor. Due to this reason, heterogeneous processors are being used for a wide variety of algorithms with considerable performance gains. In this paper, we have used state-of-the-art heterogeneous architecture namely, Cell Broadband Engine (Cell BE) to speed up the cGA+LS algorithm to solve CVRP.

Cell BE is an heterogeneous multi-core chip designed by STI (STI stands for Sony, Toshiba & IBM, alliance). Cell BE processor comprises of a 64-bit Power Processor Element (PPE) core and co-processing elements called Synergistic Processor Elements (SPE). SPEs have 256 Kb of local storage each for instructions and data. Cell BE processor is architected for multimedia applications with regular processing requirements. However, Cell BE shows significant speedups even for the problems with non-uniform memory access [14,15]. GAs are well known for their large memory requirements and non-uniform memory access patterns. These two factors call for a careful design to take maximum advantage of the Cell BE hardware.

In this paper, we propose a unique asynchronous approach for implementation of cGA+LS over Cell BE. LS, being mandatory to solve a large variety of optimization problems using GAs, is usually the most computationally intensive part in GA+LS hybrid algorithms; moreover, LS has relatively small memory requirements. These two properties of LS makes them ideal candidate to be implemented in parallel over SPEs. cGAs having large memory footprint are not so computationally intensive and are therefore more suitable for serial implementation over PPE. In our implementation, PPE runs the cGA and also act as a controller for the SPEs in an indirect manner. In the normal cases, LS is applied to all the individuals in a population. In this paper, we discuss that for large problems, instead of performing LS on all the individuals, we can select the individuals on which local search is performed based on some criteria. This results in a considerable reduction in the execution time with a negligible deterioration in the results quality. Our implementation over Cell BE is based on native Cell SDK (Ver 3.0).

The main motivation behind this paper is to harness the computational power of Cell BE, to reduce the execution time for solving existing CVRP benchmark problems and to solve CVRP problems of sizes larger than the ones attempted in the CVRP literature. To get better results for the benchmark problems than the one known in literature is not the motivation behind this paper. In most of the real life cases total execution time is the main limiting factor, and the user wants to gain

most out of the time available at hand. We claim that using the proposed method the user can make a better use of his time and approach near optimal solution. We

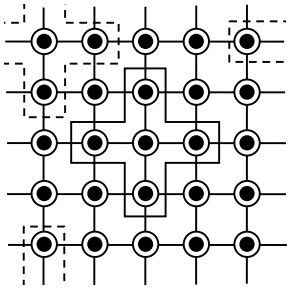

Figure 1. Population arrangement (cGA)

```
1 : Initialize cGA(InputParams);
2 : for s←1 to MAX STEPS do
3 :       for x←1 to WIDTH do
4 :             for y←1 to HEIGHT do
5 :                   NList←Get Neighborhood(Pop(x,y));
6 :                   Parents←LocalSelect(NList);
7 :                   AuxIndiv←Recombination(Pc, Parents);
8 :                   AuxIndiv←Mutation(Pm, AuxIndiv);
9 :                   AuxIndiv←Local Search(Pl, AuxIndiv);
10:                   EvaluateFitness(aux indiv);
11:                   InsertIfBetter(Pop(x,y), AuxIndiv, AuxPop);
12:             end for
13:       end for
14:       Pop←AuxPop;
15:       UpdateStatistics(Pop);
16: end for
17: Destroy();
```

Figure 2. Pseudocode for the cGA+LS [2]

show that the proposed methods can be used to get significant speedups for the known benchmark problems. The quality of results remains comparable to any state-of-the-art CVRP solver. We apply the technique to problems containing 50-2000 customers. To the best of the author's knowledge, CVRP of size greater than 1024 customers has never been attempted before. The proposed implementation is suitable for a large variety of optimization algorithms that require a hybrid of GA+LS for their solutions. This approach enables us to solve relatively large scale problems over limited distributed memory multicore architectures like Cell BE. We believe that such architectures will become more and more common in the coming years, making similar multicore implementations of algorithms inevitable in near future. We discuss the results and their implications in detail in Sect. 6.

The paper is arranged in the following manner: Section 2 describes the CVRP in detail. Section 3 defines the cGA+LS hybrid algorithm used to solve CVRP. Section 4 explains the Cell BE architecture along with the conventional parallelization models to program this architecture. In Sect. 5, we discuss the proposed parallelization model to solve CVRP using cGA+LS over Cell BE. Section 6 shows the results obtained and compares them with the results obtained by other state-of-the-art heuristics based methods in the literature. Section 7 concludes this paper and gives some guidelines for future work.

2. Capacitated VRP

As discussed earlier CVRP is a form of VRP. It can be defined mathematically as an undirected graph: Let *G=(V,E)* be a complete undirected graph with a set of *n+1* vertices $V=\{v_0, v_1, ..., v_n\}$ and a set of edges *E*. v_0 is a central depot and $v_i (i \in 1,...,n)$ represents *n* customers, each having a non-negative demand q_i. Each edge (v_i,v_j) has a non-negative distance c_{ij} (usually Euclidean distance). The CVRP finds a set of *m* routes of minimum total distance, such that each route starts and ends at the central depot, each customer is visited exactly once, and the total demand of any route does not exceed *Q* (vehicle capacity constraint). It's important to note that *m* is also a decision variable. A solution *S* of CVRP is a set of *m* routes i.e. $(R_1, ..., R_m)$. The cost of any given route $Ri=\{v_0, v_1,..., v_{(k+1)}\}$ (where, v_0 & $v_{(k+1)}$ are central depot) can be given as:

$$Cost(R_i) = \sum_{j=0}^{k} C_{j,j+1} \quad (1)$$

The total cost of a problem solution (*S*) can be given by:

$$F_{VRP}(S) = \sum_{i=1}^{m} Cost(R_i) \quad (2)$$

In simple terms CVRP strives to find a set of routes of minimum total distance, such that each route starts and ends at the central depot, each customer is visited exactly once, and the total demand of any route does not exceed vehicle capacity constraint.

3. cGA + LS

GAs are a particular class of Evolutionary Algorithms (EAs) that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover [16]. Cellular GAs (cGAs) are a subclass of GAs with spatially structured population, i.e. the individuals can only mate with their neighboring individuals [17]. These overlapped small neighborhoods help in exploring the search space because the induced slow diffusion of solutions through the population provides a kind of exploration, while exploitation takes place inside each neighborhood by genetic operators.

In cGAs the population is structured in a 2D toroidal grid. The neighborhood always contain 5 individuals: the considered one (position(x,y)) plus the North, East, West, and South individuals. Binary tournament selection is used in the neighborhood for the first parent, while the other one is the considered the individual itself as shown in Figure 1. Use of cGA + LS to solve CVRP was first proposed by E. Alba et al. as given in Figure 2.

3.1 Representation

Representation of an individual is an important part of the GA design. We have used a representation in which each individual is a permutation of unique numbers. This kind of representation is commonly used for CVRP. The permutation contains the customers and the route delimiters. Each route starts and ends at the central depot, and a null route can be represented by putting two route delimiters together without any customer in-between. Length of an individual (chromosome length) is given by: $L_{indiv} = N + M_{max} - 1$, where L_{indiv} is chromosome length, N is number of customers, and M_{max} is the maximum number of routes allowed in the solution. Numbers from *0* to *N-1* represents the customers while the numbers $\geq N$ represents the route delimiters or route splitters. Figure 3-(a) shows the representation of an individual and Figure 3-(b) shows the actual solution.

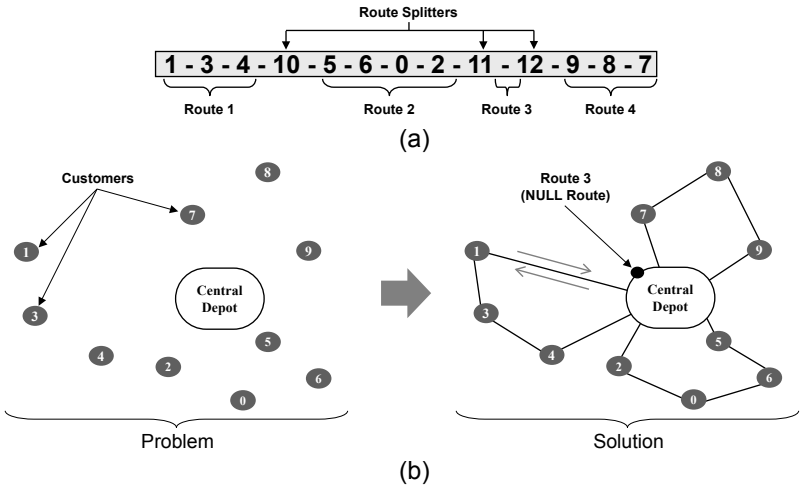

Figure 3. (a) Representation of a solution S, (b) Solution S

3.2 Crossover & Mutation

We have used the well-known ERX operator [18] as the crossover operator. For mutation we use the three mutations recommended in [2], namely *Insertion*, *Swap* and *Inversion* (see Figure 4) with equal probability. Note that mutation can change a single route or multiple routes.

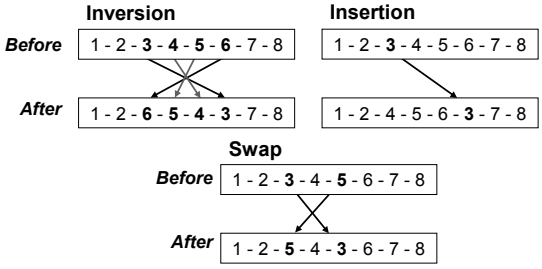

Figure 4. Types of Mutation used

3.3 Fitness Evaluation

We have used the fitness function used by most of the heuristics based CVRP solvers. The fitness is given by:

$$f_{\min}(S) = F_{VRP}(S) + \alpha \cdot overcap(S) \qquad (3)$$

Where, $F_{VRP}(S)$ is the total distance of the routes, and *overcap(S)* is the overhead of the capacity (sum of the excess of demand on routes) in a solution candidate *S*. We are using the penalty method to handle the capacity constraint of vehicles. A route exceeding the capacity of a vehicle is penalized by $\alpha \cdot overcap(S)$. α is a scaling factor and the value we have used is 10^3 as suggested by E. Alba et al. [2].

3.4 Local Search

LS is mandatory to improve convergence of GAs in case of a large variety of optimization problems. We have used two well known LS techniques namely, 2-Opt and λ-Interchange [6,7]. 2-Opt is applied within a route and λ-Interchange between two different routes. 2-Opt randomly selects two non-adjacent edges (i.e. (a, b) and (c, d)) of a single route, delete them, thus breaking the tour into two parts, and then reconnecting those parts in the other possible way (i.e. (a, c) and (b, d)). The λ-Interchange local optimization method we use is based on the interchange of all the possible combinations for up to λ customers between sets of routes. Hence, this method results in customers either being shifted from one route to another, or being exchanged between routes. It can also result in destruction of an existing route or creation of a new route. We have kept the value of $\lambda=1$ throughout the experiments. LS routine repeatedly applies 2-Opt and λ-Interchange for LS_{iter} number of times. Value of LS_{iter} is kept constant at *20* in the original cGA+LS algorithm.

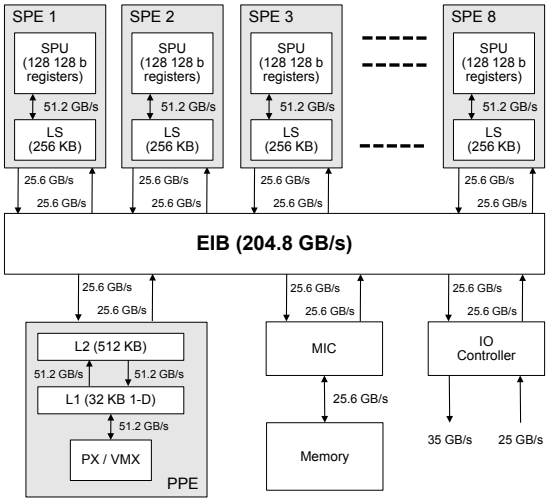

Figure 5. Architecture of Cell BE

4. Cell BE

As mentioned earlier we are using Cell BE, a state-of-the-art heterogeneous processor. The first generation Cell Broadband Engine (Cell BE) architecture has one Power Processor Element (PPE) core (two-way multithreaded) acting as the controller for the eight Synergistic Processing Element (SPEs) (SPEs are specially designed for computationally intensive jobs) as shown in Figure 5, each SPE has 256 Kbytes of local memory for data and instructions. Both PPE and SPEs support vector (SIMD) instructions. Due to the limited local memory of SPEs, Cell BE is considered to be a challenging processor to program. SPEs and PPE are connected via Element Interconnect Bus (EIB) and can communicate in several different ways. The most efficient way of transferring large amounts of data between PPE and an SPE is by using Direct Memory Access (DMA) operations. All the communication in the proposed implementation is done by this method.

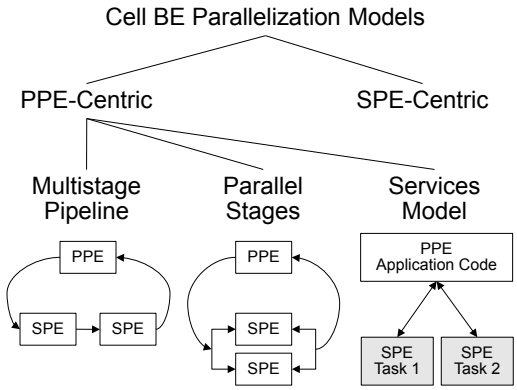

Figure 6. Cell BE Parallelization Models

4.1 Parallelization Models

There are two common parallel programming models for Cell BE as shown in Figure 6:

- *PPE Centric Model*: In PPE centric model the main application runs on the PPE, and individual tasks are offloaded to the SPEs. PPE then waits for, and coordinates, the results returning from the SPEs. This model fits an application with serial data and parallel computation. PPE centric model can be further subdivided into (1) Multistage pipeline

model, (2) Parallel stages model, and (3) Services model as shown in Figure 6.

- *SPE Centric Model*: Most of the application code is distributed among the SPEs. The PPE acts as a centralized resource manager for the SPEs. Each SPE fetches its next work item from main storage (or its own local store) when it completes its current work. In the proposed implementation, we are using SPE centric model.

4.2 Parallelization for GAs

In a typical parallel GA, the population is distributed among the available computational resources and based on some criteria individuals are exchanged with other sub populations. This model is suitable for a cluster of processors or shared memory multicore processors like AMD Opteron and Intel CoreDuo, where each processor has direct access to the main memory or distributed memory architectures with large local memory for each processing core. In case of architectures like Cell BE this is feasible only for very small problems with small populations, e.g. for a synchronous GA, CVRP involving N customers and population size P, the memory required M_{req} can be given by: $M_{req}>2 \cdot N \cdot P \cdot S_{int}$ Bytes (where S_{int} is the size taken to represent one Integer). For $N=500$, $P=1000$ and $S_{int}=4$: $M_{req}>4$ *MBytes*, when distributed among eight SPEs this comes out to be > *500 KBytes* per SPE (greater than the available local memory). Therefore, the above mentioned model (the island model) is not a feasible option to solve large instances of CVRP over Cell BE. Therefore, conventional parallel GAs are not suitable for implementation over Cell BE especially for large problems and/or large populations.

5. Proposed Implementation over Cell BE

In this paper, we discuss an asynchronous implementation of cGA+LS over Cell BE in detail. The main design considerations for the design are the limited local storage of each SPU and the communication bandwidth between the main memory and SPU local store. We discuss how the algorithm can be modified to make maximum use of the available resources.

5.1 Improvement in the LS

In the original cGA+LS algorithm [2] the local search routine applies λ-Interchange and 2-opt to each individual repeatedly for a fixed number of times LS_{iter}. LS_{iter} was kept constant at 20 in the original algorithm. In our implementation the number of LS iterations is not a constant. We fix the maximum number of iterations at 20, but in every iteration we analyze the fitness gain as compared to the last iterations. If the fitness gain is zero we stop there as we have already reached the local minima. This small modification has a significant effect on the overall execution time. The number of LS iterations for each individual decreases as the algorithm progress towards the absolute minima.

5.2 Asynchronous cGA+LS over Cell BE

As discussed in Sect. 3, LS is mandatory for solving a wide range of optimization problems using GAs. Here we propose a unique method to implement cGA+LS over Cell BE using SPE centric parallelization model (Sect. 4.1). In the proposed parallelization model, LS runs as a daemon process over all the SPEs. These threads are independent to fetch the data from the main memory. PPE on the other hand is responsible to run the cGA and to prepare the data for SPEs. Therefore, PPE although controlling SPE in some way is not acting as an explicit controller. We call it asynchronous implementation, as the cGA (running over PPE) and LS (running over SPEs) runs independent of each other and are not synchronized.

5.2.1 Memory Layout

All the communication between SPE's and PPE is done through SPE initiated DMA operations. PPE has two buffers for each SPE, one for sending the data to an SPE and the other one for receiving the data from an SPE, as shown in Figure 7. The SPE always reads the data from its respective *Out buffer* and writes the data to the respective *In buffer*. Flags are used to

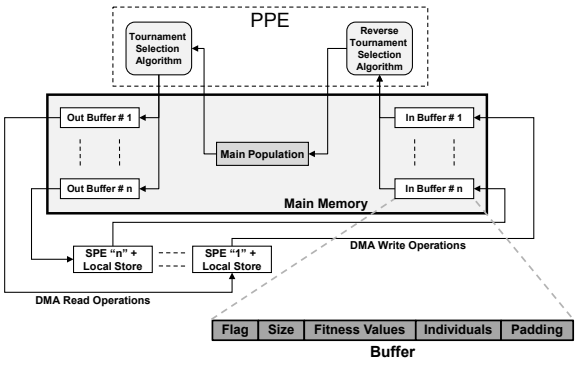

Figure 7. Memory Layout & Communication Buffers

synchronize the data access between PPE and SPE. All the communication in our implementation occurs between SPE and PPE only.

Each buffer consists of 5 fields namely *Flag*, *Size*, *Fitness_Values*, *Individuals*, and *Padding*. *Flag* is used for message passing between the PPE and SPE thread and is also used for synchronizing the data interchange between the two threads. *Flag* can be either *NEW_DATA*, *EMPTY*, or *EXIT*. *NEW_DATA* represents presence of new data, while *EMPTY* simply shows an empty buffer. *EXIT* signal is used at the termination of the program. It is a message for the SPE threads to initiate the destruction sequence. *Size* is the number of individuals on which we want to perform LS; it is kept in between 1-5 (for large instances of CVRP we take it as 1). *Fitness_Values* is a sequence of single precession fitness values of all the individuals in the buffer. *Individuals* contains the permutations of all the individuals. While, *Padding* is used to make the memory size in multiples of 128 bytes. This is preferable size for making DMA operations in case of Cell BE.

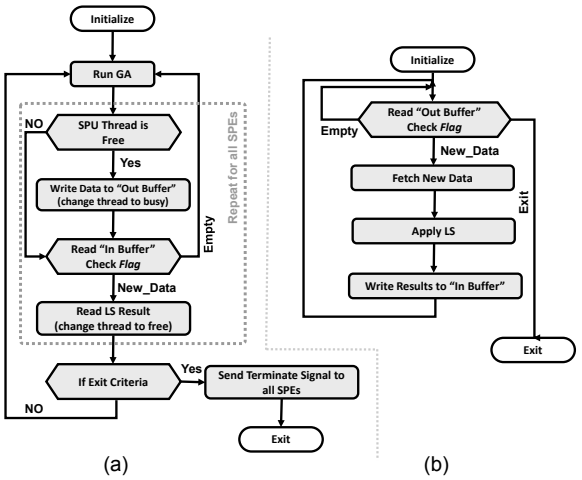

Figure 8. Controller: (a) PPE side control logic, (b) SPE side control logic (SPE uses DMA operations to perform read and write operations on "In/Out Buffers").

5.2.2 Controller Logic

Control logic for both PPE and SPE is shown in Figure 8.

- *PPE side controller*: is shown in Figure 8-(a). PPE has three main responsibilities: to run cGA, to fill the SPE buffers, and to read the results returned by the SPE's and insert them into the main population.
- *SPE side controller*: is shown in Figure 8-(b). SPE has very simple control logic. It simply checks for the new data. Whenever new data is available SPE reads the data, performs local search on it and write the data back into the main memory. SPE threads run independent of the main thread and are not directly controlled by it. However, the *Flag* of the buffer holds a control message for the SPE's.

PPE sets the value of *Flag* in all the *Out Buffers* to *EXIT* in order to signal all the SPE threads to call the destroy functions and exit. Mailboxes can also be used to signal the availability of new data but based on our experience mailboxes are slower than simple DMA operations.

5.2.3 Pseudocode over Cell BE

Figure 9 shows the Pseudocode of PPE side thread and SPE side threads separately. The Pseudocode explains the controller shown in Sect. 5.2.2 in further detail. As shown in the figure, PPE thread runs a single generation of the cGA and then checks the status of each SPE thread. If any of the SPE threads is free, PPE selects some individuals from the main population and fills the *Out Buffer* corresponding to that SPE, changes the *Flag* of the *Out Buffer* to *NEW_DATA*, and changes the status of that SPE thread to *BUSY*. PPE then checks the *Flag* of all the *In Buffers*. If new data is available PPE reads the respective *In Buffer*, insert the data in the main population, change the *Flag* to *EMPTY*, and change the status of that SPE to *FREE*. *Flag* is also used to synchronize data between the SPE's and the PPE. Therefore it should be set to *NEW_DATA* only when the entire buffer is already filled with the desired data. It's important to note that all DMA operations in our implementation are SPE initiated. PPE is only responsible to fill the respective *Out Buffers*. SPE automatically reads the data from this buffer when it is free. Similarly SPE writes the data to the *In Buffer* using DMA operations. This data is then read by the PPE using simple memory read operations.

As shown in Figure 7 the criteria for selecting individuals for LS is based on tournament selection of size T_f, while the individuals (result of LS) are inserted back in the main population using a reverse tournament selection of size T_r. In reverse tournament selection, T_r individuals are selected randomly from the main population and the individual with the lowest fitness in

the main population are replaced by the ones received from the LS thread.

We have used asynchronous population (Steady State GA) to reduce the overall memory requirements.

5.3 Simple Synchronous Parallelization

For comparison purposes we also implemented synchronous cGA+LS. This approach simply uses

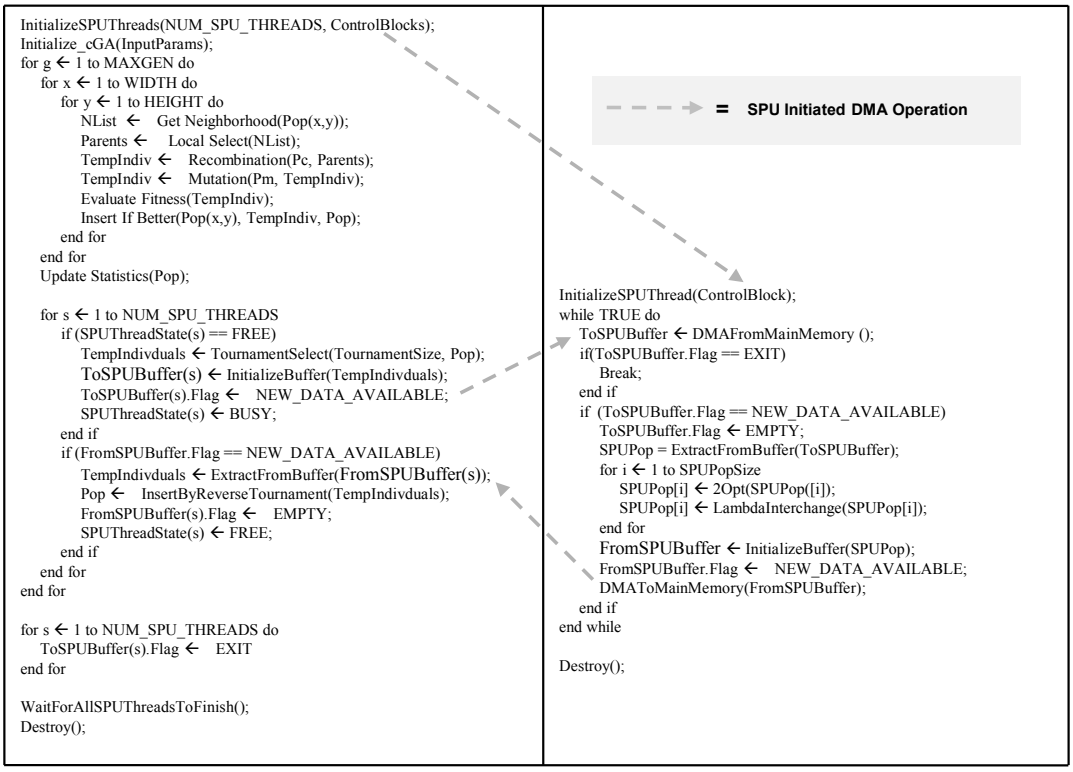

Figure 9. Pseudocode of the proposed implementation over Cell BE: (a) PPE side pseudocode, (b) SPE side Pseudocode

In our observation, asynchronous population gives almost the same results as that of synchronous population. Double buffering is not applied as the time taken by LS is a lot more than the time taken by DMA operation, therefore, adding double buffering will only add complexities to the controller without adding any considerable performance benefit.

Using the proposed model for parallelization we can easily run problems of very large size over the limited size distributed memory architectures like Cell BE. LS being mandatory and computationally very intensive (as compared to cGA) is the most suitable part of the code to be run in parallel. While, the rest of the code being very light (as compared to LS) can run on the PPE. Moreover, GAs are ideal for implementation over the single precision architecture of Cell BE, as the error in the case of GAs does not accumulate over the generations.

functional level parallelism to implement LS over the available SPUs. In synchronous approach LS is performed on all the individuals of a population and the algorithm waits for all the LS to finish before moving on to the next step. Instead of using Cell BE native code we have used Cell Superscalar (CellSs) [19] to generate parallel program from the serial code. CellSs provides a certain level of abstraction and help in writing a functional level parallel code with the addition of a few trivial annotations in the serial code. CellSs compiler converts the source code into a valid Cell BE executable. The compiler takes care of the task scheduling and data handling between the different processors of this heterogeneous architecture. Besides, it automatically implements double buffering and a locality-aware task scheduling to reduce the overhead of data transfers. Figure 10 shows the annotations that we have used to convert the serial cGA+LS code into a

functional level parallel cGA+LS code. Note that CellSs cannot maintain the state of the threads running over SPEs and hence is not useful for the asynchronous implementation discussed in the last section.

with Cell BE intrinsics. This native Cell code is compatible with Cell SDK 3.0.

We have kept the conditions constant throughout the experiments. The value of α is kept constant at 10^3, and value of λ is kept at 1 throughout the experiments.

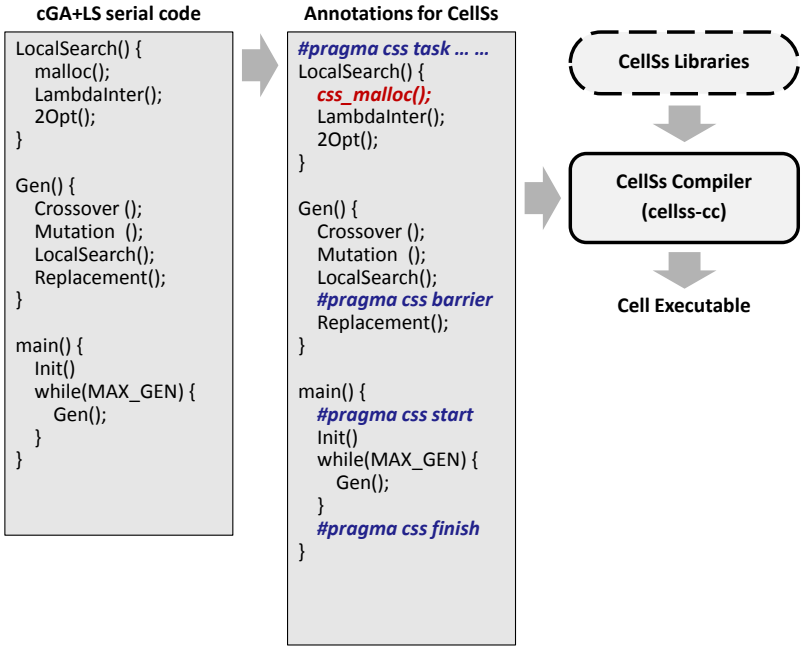

Figure 10. CellSs Annotations

6. Results

In this section, we have compared the results obtained by the proposed method with results obtained by other state-of-the-art metaheuristics based solvers in CVRP literature. We also compare the asynchronous implementation of CVRP over Cell BE with the synchronous implementation.

6.1 Environment & Benchmarks

All the results presented here were collected on a Sony PS3™ (Play Station 3) with Fedora Core 7. PS3 has Cell BE as its processor, with 256MB of main memory as compared to 1GB in a normal Cell BE configuration. Moreover, only seven of the eight SPEs are available in case of PS3. Out of these seven, one is used by the OS; therefore we are left with only six SPEs to work with. However, the same application should give even better results if executed over a normal Cell BE processor with 8 SPEs. We have implemented the proposed algorithm using plain C

Value of maximum numbers of LS iterations LS_{iter} is kept constant at 20. The population size of the cGA is also kept constant at 100. For stopping criteria, the algorithm stops if the required value is achieved which is usually the best known fitness with a fixed error margin. Algorithm also stops if the fitness value of the population does not change over the last 10 generations. Value of T_r and T_f (as given in Sect. 5.2.3) are kept constant at 4. Value of mutation probability P_m=0.85 and value of crossover probability P_c=0.65 throughout the experiment as recommended by E. Alba et al. [2]. All the results presented in this paper are average of at least 10 independent runs under similar conditions.

The test cases used in the results are taken from the famous CVRP benchmarks problems namely Christofides benchmarks [20] (represented by *C* in Table 1), and Golden benchmarks [21] (represented by *G* in Table 1). While, the benchmark problems represented by *M* are our own benchmark problems. As to the best of author's knowledge CVRP benchmarks of large size do not exist in literature, therefore, we have

created new benchmarks of sizes starting from 500 customers to 2000 customers.

6.2 Empirical Results & Discussion

In any real metaheuristics based implementation there is always a tradeoff between the quality of the solution and the total execution time. This means that a slight increase in solution quality may require a considerable increase in the total execution time. This type of tradeoff is more visible in real-life situations where we have strict time constraints. In most of the real life cases the goal is to get the best possible solution within the total available time at hand. Therefore, for a fair comparison between different algorithms, we either have to keep the quality or the execution time as a constant. As time is the limiting factor in most of the cases so it will only be fair to keep the time as a constant and observe the quality of the solution obtained by different algorithms in that fixed amount of time.

Table 1 shows the quality of solutions obtained by using different algorithms, while keeping the total execution time as a constant T_c. Where, T_c is the time taken by asynchronous cGA+LS algorithm (proposed method) to converge to a solution. We have compared the results obtained by the proposed asynchronous implementation with the synchronous implementation over Cell BE and other state-of-the-art metaheuristics based algorithms. Note that in the proposed asynchronous implementation of cGA+LS over Cell BE, LS is only applied to the individuals selected using a special criterion (see Sect. 5.2.3 for details). It is clear from the table that asynchronous approach outperforms every other method for all test cases except for *C1* (which is a relatively small instance of CVRP). Therefore, we argue that it is not important to apply LS to all the individuals when the target is not to achieve the absolute best result, but to achieve the best possible result in the given amount time.

Table 1. Time/Quality of the solutions obtained by different algorithms over different computing environments. Time is kept constant at T_c

Instances	n	T_c (Seconds)	Asynchronous *C	Synchronous *C	cGA+LS *1	E. Alba et al. *2
C1	50	8	531.25	**524.61**	**524.61**	**524.61**
C2	75	16	**858.2**	919.42	985.12	1021.23
C3	100	35	**832.71**	913.59	1026.61	971.48
C5	199	88	**1364.21**	1612.42	1982.58	2100.84
G12	483	1485	**1198.23**	1314.65	1492.27	1682.01
M1	500	1823	**5598.23**	5715.81	5828.54	6125.93
M2	750	2341	**8032.12**	8389.04	8525.18	9294.29
M3	1000	3120	**10254.19**	10396.1	10749.56	12295.58
M4	2000	3901	**21857.32**	22431.98	27321.31	25153.95

*C. 3.2GHz Cell BE (PS3), Implemented by the authors
*1. 3 GHz PIV, Implemented by the authors
*2. 2.4 GHz PC, E. Alba (2004)

Proposed Method
Implemented by the authors for comparison purposes

Figure 11 shows the deviation ($\Delta Solution$) of the solutions obtained by different methods as compared to the ones obtained by asynchronous algorithm:

$$\Delta Solution = \frac{|Solution - BestSolution|}{BestSolution} * 100 \qquad (4)$$

BestSolution in the above equation does not mean the absolute best solution, rather it is the best solution obtained in T_c amount of time by any of the algorithm shown in Table 1. As it is clear from the figure,

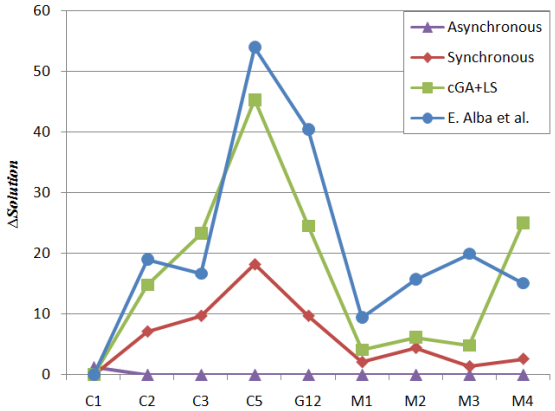

Figure 11. Quality of solutions ($\Delta Solution$)

Δ*Solution* of asynchronous algorithm is almost always 0. This supports our argument that it is not necessary to apply LS to all the individuals in a case where we have limited amount of time to reach a solution.

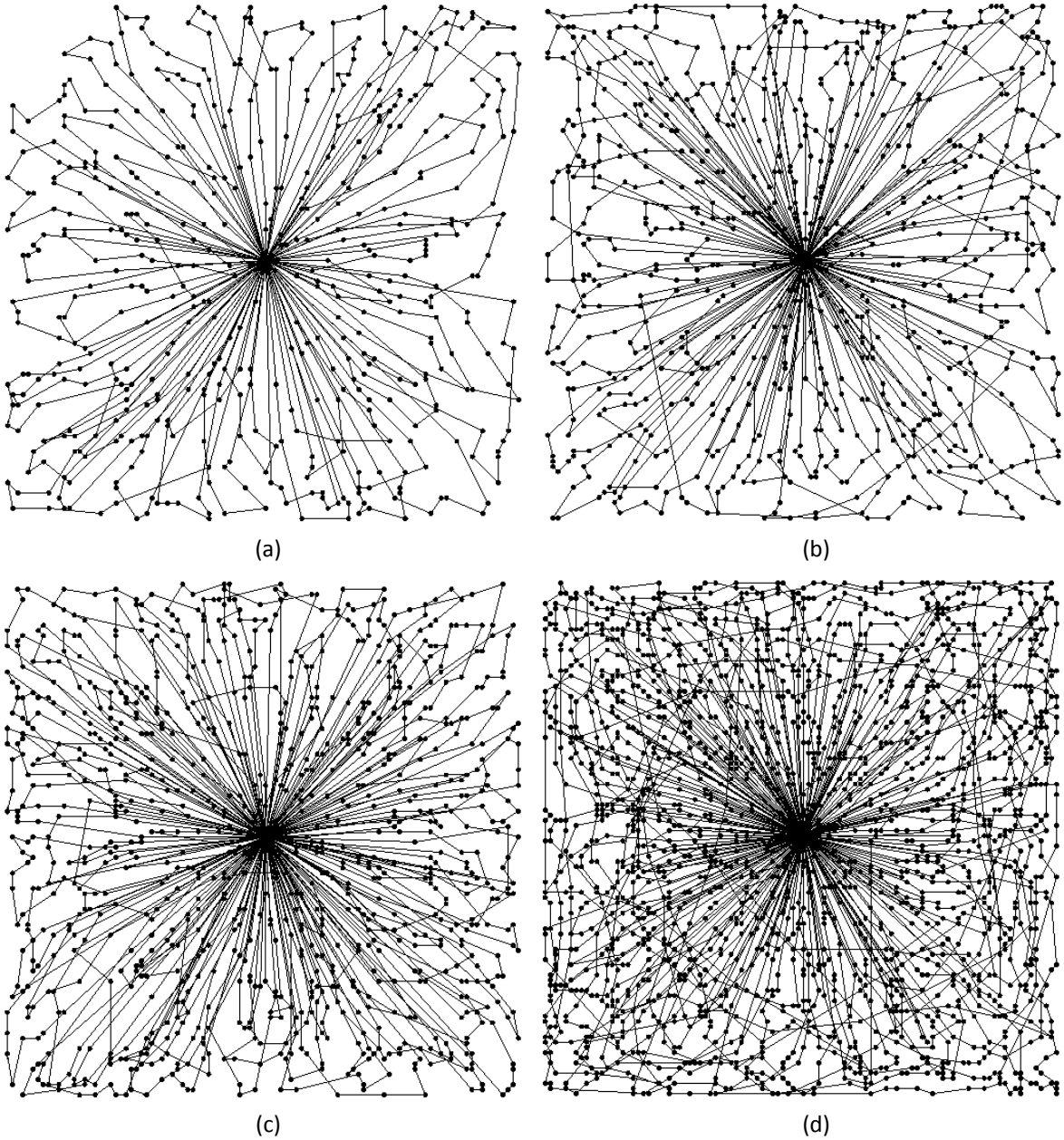

Figure 12. Snapshots of the solution for: (a) M1 benchmark problem (500 customers), (b) M2 benchmark problem (750 customers), (c) M3 benchmark problem (1000 customers), (d) M4 benchmark problem (2000 customers).

We conclude that in a limited amount of time asynchronous method gives better solution as compared to other algorithms. Such speed-up is necessary for very large scale CVRP problems which can take hours and even days to evaluate using the conventional methods of solving CVRP. We have attempted CVRP problems involving more than 500 customers. As these benchmarks are new and we have solved them for the first time; therefore, we give a snapshot of the solutions obtained by the proposed implementation in Figure 12. This figure gives a rough idea about the quality of the solutions obtained by the

proposed asynchronous implementation of cGA+LS over Cell BE.

7. Conclusions & Future Work

CVRP is far more complicated than solving multiple Traveling Salesman Problems (TSPs) in parallel. This is due to the fact that in case of CVRP the customers can move within the routes and there are further constraints not present in a simple TSP.

In this paper, we have presented a unique method for solving CVRP using cGA+LS over Cell BE like architectures. This method can be used to solve any kind of optimization problem that uses GA+LS hybrid pair for its solution. As LS usually takes most of the execution time, therefore, parallelization of LS can considerably reduce the total execution time without disturbing the result quality. We argue that implementation of parallel GAs over Cell BE in their crude form is only feasible for problems with smaller population sizes and shorter chromosome lengths.

We proposed an asynchronous algorithm that does not apply LS to all the individuals in a population. Asynchronous implementation is more feasible in real world problems where the total execution time is usually limited. Therefore, applying LS to all the individuals of a population may not be a good idea in all the cases.

We used CellSs to implement synchronous cGA+LS over Cell BE and found it to be a very efficient way to introduce functional level parallelism in a serial code.

As a future work, an exact analysis of number of local searches performed and its impact on the solution quality would be a good contribution. Number of local searches performed can be treated as a variable instead of a constant and can be increased/decreased with the increase in generations. To analyze the feasibility of using SIMD instructions for the LS algorithm is another good line of work.

8. Acknowledgements

We would like to thank Bernabé Dorronsoro for his help, regarding the use of Cellular Genetic Algorithm to solve Capacitated Vehicle Routing Problem. We would also like to thank Isaac Jurado and Rosa M. Badia for their help in using CellSs. We also thank the anonymous users who replied to our queries on the mailing lists related to CellBE.

9. References

[1] P. Toth, D. Vigo, The vehicle routing problem. Monographs on Discrete Mathematics and Applications. SIAM 2001.

[2] E. Alba, B. Dorronsoro, Solving the vehicle routing problem by using cellular genetic algorithms. EvoCOP, Lecture Notes in Computer Science, vol. 3004, Gottlieb J, Raidl G (eds.), Springer, 2004; pp 11–20.

[3] F. Glover, M. Laguna, Tabu search. Modern Heuristic Techniques for Combinatorial Problems, Reeves C (ed.), Blackwell Scientific Publishing: Oxford, England, 1993.

[4] S. Kirkpatrick, C. Gelatt, M. Vecchi, Optimization by simulated annealing. Science, Number 4598, 13 May 1983; pp 671–680.

[5] D. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional, 1989.

[6] G. Croes, A method for solving traveling salesman problems. Operations Research 6, 1958; 791–812.

[7] I. Osman, Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Ann. Oper. Res. 1993; 41(1-4): pp 421–451.

[8] E. Alba, B. Dorronsoro, Computing nine new best-so-far solutions for capacitated vrp with a cellular genetic algorithm. Information Processing Letters. 98(6): June 2006; pp 225–230.

[9] J. Berger, M. Barkaoui, A hybrid genetic algorithm for the capacitated vehicle routing problem. GECCO Lecture Notes in Computer Science, vol. 2723, Springer, 2003.

[10] Y. Nagata, Edge assembly crossover for the capacitated vehicle routing problem. EvoCOP, Lecture Notes in Computer Science, vol. 4446, Cotta C, van Hemert J (eds.), Springer, 2007; pp 142–153.

[11] Y. Rochat, E. Taillard, Probabilistic diversification and intensification in local search for vehicle routing. Journal of Heuristics 1995; pp 147–167.

[12] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams, The landscape of parallel computing research: A view from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley Dec 2006.

[13] S. Che, M. Boyer, J. Meng, D. Tarjan, JW. Sheaffer, K. Skadron, A performance study of general-purpose applications on graphics processors using CUDA. Journal of Parallel and Distributed Computing, Vol. 68, No. 10. (October 2008), pp. 1370-1380.

[14] J. Kurzak, J. Dongarra, Implementation of mixed precision in solving systems of linear equations on the cell processor: Research articles. Concurrency and Computation: Practice and Experience 2007; 19(10): pp 1371–1385..

[15] M. Xu, P. Thulasiraman, Parallel algorithm design and performance evaluation of fdtd on 3 different architectures: Cluster, homogeneous multicore and Cell/B.E. Proceedings of HPCC '08: 10th IEEE International Conference on High Performance Computing and Communications, Dalian, China, 2008; 174–181.

[16] D. Petrowski, S. Taillard, Metaheuristics for Hard Optimization: Methods and Case Studies, chap. Evolutionary Algorithms. Springer, 2006; pp 75–92.
[17] E. Alba, B. Dorronsoro, Cellular Genetic Algorithms, Operations Research/Computer Science Interfaces Series, vol. 42. Springer, 2008.
[18] W. Darrell, T. Starkweather, D. Fuquay, Scheduling problems and traveling salesman: The genetic edge recombination operator. International Conference on Genetic Algorithms, 1989; 133–140.
[19] P. Bellens, J. Perez, R. Badia, J. Labarta, Cellss: a programming model for the cell be architecture. Supercomputing, 2006. SC '06. Proceedings of the ACM/IEEE SC 2006 Conference Nov 2006.
[20] N. Christofides, A. Mingozzi, P. Toth, Combinatorial Optimization, chap. The vehicle routing problem. John Wiley & Sons, 1979.
[21] B. Golden, E. Wasil, J. Kelly, I. Chao. Fleet Management and Logistics, chap. Metaheuristics in vehicle routing. Kluwer Academic Publishers: Boston, 1998.