Optimization of shared-memory multicore systems using Game Theory and Genetic Algorithms on Cellular Automata lattices.

Michail-Antisthenis I. Tsompanas
Department of Electrical and Computer Engineering
Democritus University of Thrace, DUTH
Xanthi, Greece
mtsompan@ee.duth.gr

Christoforos Kachris
Department of Electrical and Computer Engineering
Democritus University of Thrace, DUTH
Xanthi, Greece
ckachris@ee.duth.gr

Georgios Ch. Sirakoulis
Department of Electrical and Computer Engineering
Democritus University of Thrace, DUTH
Xanthi, Greece
gsirak@ee.duth.gr

Abstract— **A main problem in multi-core architectures is the run-time management and the allocation of shared resources, such as the shared memory. This paper presents a model of the memory resources allocation, in specific the on-chip shared memory, in order to achieve higher performance based on the basic concepts of game theory and the iterated spatial prisoner's dilemma game, with the help of adaptive computational tools like cellular automata and genetic algorithms. The paper evaluates the proposed scheme using multi-core applications that are based on the MapReduce framework. The performance evaluation and the simulation results show that the allocation of shared resources based on game theory and genetic algorithms, can improve, significantly, the overall performance of the multi-core architectures.**

Keywords— cellular automata, genetic algorithm, game theory, cache memory, multicore systems

I. INTRODUCTION

Throughout history, improvements on performance of microprocessors were achieved by increasing the frequency of processors, and by increasing the amount of parallelism. In recent years, it seems that existing techniques for increasing instruction-level parallelism can hardly deliver marginal performance improvements that track Moore's Law due to energy, heat, and wire delay issues [1]. As a result, microprocessor industry focuses to thread-level parallelism (TLP) by designing chips with multiple processors, known as multicore processors. By extracting higher-level TLP on multicore systems, performance will further improve, while managing the technology issues faced by increasing the performance of conventional single-core designs. Computer companies invest on that direction, by rapidly increasing the number of processing cores per chip. Using multicore processors will rely on parallel software to achieve continuing exponential performance gains. The majority of the commercial parallel software relies on the shared-memory programming model, in which all processors access the same physical address space. Although processors logically access the same memory, on-chip cache hierarchies are crucial to achieve fast performance for the majority of memory references made by processors. Thus a key problem of shared-memory multicore processors is providing to each core, a consistent view of memory with various cache hierarchies.

Therefore, there is a lot of interest in ways to achieve better memory performance. Some recent work examines multiple memory controllers (MCs) in a multi-core setting. Vantrease et al. [13] discuss the interaction of MCs with the on-chip network traffic and propose physical layouts for on-chip MCs to reduce network traffic and minimize channel load. The Tile64 processor [14] employs multiple MCs on a single chip, accessible to every core via a specialized on-chip network. Tile microprocessor [14] was one of the first processors to use multiple (four) on-chip MCs. More recently, Abts et al. [2] explored multiple MC placement on a single chip-multiprocessor so as to minimize on-chip traffic and channel load. None of the above considers intelligently allocating data and load across multiple MCs. Kim et al. propose ATLAS [9], a memory scheduling algorithm that improves system throughput without requiring significant coordination between N on-chip memory controllers. Mars et al. [10] halt low priority processes when contention is detected. Herdrich et al. [6] analyze the effectiveness of frequency scaling and clock modulation to reduce shared resource contention. Awasthi et al. [3] show that data migration and adaptive memory allocation can be used to reduce memory controller overhead in systems with multiple memory controllers.

The main concept here is the struggle of an individual core to acquire some resources that are in common use, and so may be useful for other cores too. So, what would be the best strategy for that core; to acquire the resources for the time needed, regardless the other cores' needs? To acquire the resources for some fragments of time and let the others use it too, but making considerably longer to finish a task? Consequently, game theory emerges and inspired by the local interaction of the cores, each core of the under study processor, can be represented as a Cellular Automaton (CA) cell and, more specifically, as a player in a community with a predefined number of CA neighbors, who will conflict for the occupancy and procession of resources [12]. This concept gives the opportunity to test various resources distribution strategies against each other and, under some certain circumstances,

decide which configuration provides higher performance for the system. Taking into account, the results from previous works [12], it is safe to admit that dynamic strategies achieve better scores, thus they are more efficient. Taking this fact into consideration, a Genetic Algorithm (GA) is the best candidate, because GAs, by definition, adapt to their environment, as they are search procedures that mimic the mechanics of genetics and natural selection. In other words, the usage of GAs, grants greater variety of choices for the CA rules, the behavior of each cell becomes more adaptive to real life needs of modern memories, it is more difficult to drive some cells into resource-starvation and the system becomes more balanced, as it uses the most out of the memory without leaving cells without resources.

Finally, in order to compare the behavior of the proposed here model with the performance of real systems, metrics of a parallel algorithm running on a multicore shared-memory microprocessor were obtained. Phoenix [7] was used, an implementation of MapReduce for shared-memory systems that includes a programming API and an efficient runtime system. These metrics were analyzed in order to depict the Last Level Cache (LLC) references, a unit that directly correlates with the execution time of an application, due to high latencies. Using results from the profiling, for adjusting the parameters of the model, provides the right circumstances to simulate the behavior of the system. Finally, the strategies of the players constituting the community were altered, introducing GA strategy players, in order to compare the overall performance of the system by using dynamic strategies. It should be noticed that bearing in mind the alteration of memory system behavior at every moment depending on different requests and constraints, the implementation of GAs provide a adaptive environment easy to follow up the requested strategy changes. The comparison of the results, send out an optimistic message for the increasing performance of multi-core systems, using Game Theory and GA in CA, as a speed-up is estimated by the proposed model by different task scheduling of the application.

This paper is organized as follows: In Section II, all the necessary background on Game Theory, Cellular Automata and Genetic Algorithms is provided, respectively. Section III presents the simulation principles of the proposed model, whereas in Section IV, the real multicore system configuration, the metrics used and the connection between model and reality are thoroughly described. In the end, the conclusions drawn and future work are presented in Section V.

II. Game Theory, Cellular Automata and Genetic Algorithms

Game theory can be defined as the study of mathematical models of conflict and cooperation between intelligent rational decision-makers. Game theory provides general mathematical techniques for analyzing situations in which two or more individuals make decisions that will influence one another's welfare [8]. On the other hand, CA are models of physical systems, where space and time are discrete and most important interactions are local [21]. As a result, the application of game theory to CA can be proved very promising for the development of useful modeling tools.

Game theory provides mathematical tools to model, structure and analyze interactive scenarios. The players in a game may be, for example, competing firms or buyers and sellers on the internet. The language and concepts of game theory are widely used in economics, political science, biology, and computer science, to name just a few disciplines. Game theory helps to understand effects of interaction that seem puzzling at first. Other insights come from the way of looking at interactive situations. Game theory treats players equally and recommends to each player how to play well, given what the other players do [22]. This mindset is useful in strategic questions of management, because "you put yourself in your opponent's shoes" [4]. Some of the "games" studied by game theory are the Hawk-Dove game, the Tragedy of the Commons, Stag hunt and the Prisoner's Dilemma [28].

Game theory is fascinating as a topic because of its diverse applications. The ideas of game theory started with mathematicians, most notably the outstanding mathematician John von Neumann. In the 1950s, a group of young researchers in mathematics at Princeton developed game theory further, among them John Nash, Harold Kuhn and Lloyd Shapley, and these pioneers can still, over 50 years later, be met at conferences on game theory. Most research in game theory is now done by economists and other social scientists [4]. However, game theory models found applications in the field of technology too [22 – 24, 26].

A CA consists of a regular uniform n-dimensional lattice (or array), usually of infinite extent. At each site of the lattice (cell) a physical quantity takes on values. This physical quantity is the global state of the CA, and the value of this quantity at each site is its local state. The states at each cell are updated simultaneously at discrete time steps, based on the states in their neighborhood at the preceding time step. The algorithm used to compute the next cell state is referred to as the CA local rule. Usually the same local rule applies to all cells of the CA. A CA is characterized by five properties [16, 17]:

1. the number of spatial dimensions (n);
2. the width of each side of the array (w). w_j is the width of the j^{th} side of the array, where $j = 1, 2, 3, \ldots, n$;
3. the width of the neighborhood of the cell (d). d_j is the width of the neighborhood along the j^{th} side of the array;
4. the states of the CA cells;
5. the CA rule, which is an arbitrary function F.

The state of the $\vec{r}$ cell, at time step (t+1), is computed according to F. F is a function of the state of this cell at time step (t) and the states of the cells in its neighborhood at time step (t). In the above definition, the function F is identical for all sites and it is applied simultaneously to each of them, leading to a synchronous dynamics. It is important to notice that the rule is homogeneous, i.e. it does not depend explicitly on the cell position $\vec{r}$ .However, spatial inhomogeneities can be introduced by having some cell state $C_j(\vec{r})$ systematically at 1, in some given locations of the lattice, to mark particular cells for which a different rule applies. Furthermore, the new state at time t+1 is only a function of the previous state at time t. It is sometimes necessary to have a longer memory and introduce a dependence on the states at times t-1, t-2, …, t-k.

Such a situation is already included in the definition, if one keeps a copy of the previous state in the current state.

The neighborhood of cell $\vec{r}$ is the spatial region in which a cell needs to search in its vicinity. In principle, there is no restriction on the size of the neighborhood, except that it is the same for all cells. However, in practice, it is often made up of adjacent cells only. For two–dimensional CA, two neighborhoods of range *r* are often considered: Von Neumann neighborhood, defined as follows:

$$N^{N}_{(x0,y0)} = \{(x, y) : | x - x_0 | + | y - y_0 | \leq r\} \quad (1)$$

and Moore neighborhood, which can be described by the following equation:

$$N^{M}_{(x0,y0)} = \{(x, y) : | x - x_0 | \leq r, | y - y_0 | \leq r\} \quad (2)$$

In practice, when simulating a CA rule, it is impossible to deal with an infinite lattice. The system must be finite and have boundaries. Clearly, a site belonging to the lattice boundary does not have the same neighborhood as other internal sites. In order to define the behavior of these sites, the neighborhood is extending for the sites at the boundary. Extending the neighborhood leads to various types of boundary conditions such as periodic (or cyclic), fixed, adiabatic or reflection [18].

CA have sufficient expressive dynamics to represent phenomena of arbitrary complexity and at the same time can be simulated exactly by digital computers, because of their intrinsic discreteness, i.e. the topology of the simulated object is reproduced in the simulating device. The CA approach is consistent with the modern notion of unified space–time. In computer science, space corresponds to memory and time to processing unit. In CA, memory (CA cell state) and processing unit (CA local rule) are inseparably related to a CA cell [19]. In addition, algorithms based on CA run quickly on digital computers [20]. Models based on CA lead to algorithms which are fast when implemented on serial computers, because they exploit the inherent parallelism of the CA structure [19, 20, 12].

Using principles of Prisoner's Dilemma (PD), a very popular instance of game theory [28], on cellular automata lattices seems as a powerful tool to simulate the dynamics of conflict over shared resources. In the original two-person, one-shot game the players have two options, cooperation (*C*) and defection (*D*). Each player's payoff depends on his choice and the choice of his opponent. A typical payoff matrix is illustrated in Table I.

To be more specific, for mutual cooperation each player receives the reward *R*, two defectors receive the punishment *P*, while a cooperator and defector receive the sucker's payoff *S* and the temptation *T* (to choose defection), respectively. For the PD game the payoffs satisfy the ranking:

$$T > R > P > S \quad (3)$$

TABLE I. PRISONER'S DILEMMA PAYOFFS

		B	
		Cooperate (C)	Defect (D)
A	Cooperate (C)	3 3	5 0
	Defect (D)	0 5	1 1

According to the assumption of the traditional game theory, players make a rational decision to maximize their own income. Consequently, they should choose defection independently from the other player's decision. For the iterated PD game an additional constraint is assumed to provide the highest total income for mutual cooperation [11]:

$$T + S < 2R \quad (4)$$

The question then, is to find under which conditions the cooperation emerges when this game is played repeatedly [5]. The objective for each player is to collect the largest amount of points which now requires that some overall strategy be adopted. Furthermore, a spatial component can be added to the game so that it is played on a square grid against multiple partners.

GAs are search procedures that mimic the mechanics of genetics and natural selection [25]. They provide robust searching capabilities in complex problem solution spaces. A possible solution, called a "chromosome", contains smaller building blocks referred to as "genes". A set of chromosomes is referred to as a "population". A target function (whose value we seek to optimise) provides the mechanism for evaluating the solution represented by each chromosome. This function is referred to as a "fitness function", and the value assigned to each chromosome, using the fitness function, is the "chromosome fitness" [27]. The chromosome with the highest fitness represents the optimal solution.

GAs start with a randomly chosen initial population. The initial population comprises a relatively small number of chromosomes. They produce the next generation by performing three genetic operations on the initial population, namely selection, crossover and mutation. Selection is a process that selects superior chromosomes, i.e. those with the most optimal fitness function values, which will survive to the next generation, and also selects inferior chromosomes, which will perish. A simple selection strategy is to allow a number of the fittest chromosomes to survive, and to discard the less fit ones. In every generation, crossover generates offspring by exchanging genetic material between pairs of highly fitted chromosomes. Crossover operation will be utilised, in order to introduce novel combinations of the genetic material. After crossover, chromosomes are subjected to mutation. A random gene of a random chromosome is selected and mutated, by changing its value from 0 to 1, or vice versa. Mutation operation increases the population variability, thus helping to prevent irrecoverable loss of potentially important information concerning the solutions of the problem at hand. The iteration

stops when an arbitrarily acceptable solution is reached or after a given number of generations.

III. SIMULATION

The proposed model is based on a simulation environment, which generates a Spatial Iterated Prisoner's Dilemma game on a CA lattice. This simulation environment has been developed using MATLAB. Considering the cores of a homogeneous multicore system to be identical, they are represented by players - CA cells, which are placed on a square grid. Here, the Moore neighborhood was used. Furthermore, periodic and adiabatic boundary conditions were selected in order to simulate different configurations.

The equivalent of the model to a real-life multicore processor's behavior is the following. When a core is assigned with high computation load, or it "defects", it is natural that it will require more shared memory resources in order to complete it. Moreover, it was assumed that if there is no available memory resources, the core, in need of them, must wait until some resources are accessible, decreasing the performance of the system. On the other hand, if a core "cooperates" or is assigned with lower computation load, it does not need too many resources, to accomplish a task. Consequently, in a game of two cores, if both "defect" or need resources, the payoff is 1/1, simulating the low performance of the system, consisting of two "greedy" cores. However, if neither needs excessive resources (cooperation) there is no bottleneck experienced, and as a result the sum of the collected score of both players is the highest possible (3/3). Finally if one needs resources while the other does not, the performance of the system is higher than the first example, but lower than the latter (5/0).

To complete one round of the game, each player conflicts with the players composing its neighborhood. The result of this conflict, which is summed to the results gained from earlier rounds, is subject to the moves each player does and shown in Table I. If *S(n)* is the total score a player has achieved until round n and P_1, P_2, P_3 and P_4 the payoffs from the interaction between the player and each one of its neighbors on that round, then its total score on the round *n+1* will be:

$$S(n+1) = S(n) + P_1 + P_2 + P_3 + P_4 \quad (5)$$

The choice of the move made for every round by each core of the processor, i.e. player, is determined by the strategy (local rule) of every CA cell. Note that for the rest of the manuscript the term player, accordant with game theory terminology, will be used.

Furthermore, altering the initialization parameters of the proposed model, results in simulating systems with different configurations. These parameters are the number of players forming the CA grid, the local rule or strategy followed by each player, the type of the CA neighborhood and the boundary conditions of the grid.

The strategies that a player can adopt are separated into the dynamic and the static ones. The static strategies are the defective and the cooperative. Defective strategy is followed by a player that always defects. Cooperative strategy is followed by a player that always cooperates. The dynamic strategies are the random, Tit-for-Tat and Pavlov strategy. Here, only random strategy was used, which dictates the player to choose with a specific possibility if he will defect or cooperate. Moreover, Tit-for-Tat is followed by a player that cooperates on the first round and then replicates the moves made by other players and Pavlov is followed by a player that repeats its former choice whenever it earns a high payoff like 5 or 3 and switches that choice whenever it earns a low payoff like 1 or 0.

It must be pointed out that the initial model proposed in [12], was altered in order to simulate the real multicore system that was used for profiling. The model used here, was modified from the initial in terms of the type of the CA neighborhood, the size of the CA grid and the boundary conditions. Furthermore, the possibilities of the Random strategy have been calibrated based on data from the profiling of Phoenix from its application to a multi-core processing system.

Moreover a new strategy is introduced in this paper that uses a genetic algorithm (GA). The algorithm collects data from every three rounds to decide the moves the player will do for the next three rounds. It should be noted that different numbers of rounds for the GA application were tested several times and for different tactics in order to provide more fruitful results in correspondence to the requested system performance. In correspondence, the simulation has been initialized to run for three rounds, in order to make it easier for the GA players to choose the best chromosomes. For the first three rounds the player cooperates with all his neighbors. After that and at the end of every set of three rounds, the selection phase is initiated. The fitness proportionate selection method is used in order to find two superior chromosomes, from the available, which are the moves the players' neighbors did in these three games. Using the philosophy of that method, the score of every neighbor is normalized such that they sum up to 1, and then the chromosomes are selected with probability similar to the value of the normalized scores. As a result a strategy of a player with a small score can also be selected.

Then, the algorithm decides the moves that the player will follow in the next set of three games. This is done by the Uniform Crossover method [29]. This method uses for the offspring one of the parents' gene with probability of 50%. Moreover, although, mutation is a basic genetic operand in GAs, this implementation limits its usage in less than 1%. This choice was made, due to the fact that applying mutation in every turn to a chromosome consisting of three bits, will alter the chromosome by 30%, thus leading to huge alterations of the GA proposed best solutions. For further understanding, the outline of the genetic crossover strategy algorithm is presented:

Chromosome="111" (The first three moves of the player is to cooperate)
FOR 200 game rounds
IF (first round in a set of three)
Use 1st gene-bit of the present chromosome
ELSE IF (second round in a set of three)
Use 2nd gene-bit of the present chromosome
ELSE IF (third round in a set of three)
Use 3rd gene-bit of the present chromosome
END IF
IF (third round in a set of three)

Select with proper probability two chromosomes (the last 3 moves of two neigbors)
Crossover the chromosomes (use with 50% chance one of each chromosomes' gene-bits)
Save the final chromosome (consisting three moves-bits for the next three games)
END IF
END FOR

IV. Real Multicore System Configuration and Comparison with the Results of the Model

In order to obtain metrics of a multicore system, Phoenix [7] MapReduce runtime was used. MapReduce framework is commonly used in distributed systems such as data centers or HPC and offers simplicity and scalability for the parallel programmers. Phoenix is a multi-core implementation of the MapReduce framework and it uses threads to spawn parallel Map or Reduce tasks. It also uses shared-memory buffers to facilitate communication. The runtime schedules tasks dynamically across the available processors. Phoenix was run on a shared-memory system described in Table II.

Using the system described in Table II does not limit the potentials of the proposed model. Note that the model can be altered to simulate a high number of players, different type of neighborhood and even another payoff table. Altering these parameters can capture different trade-offs appearing in different types of microprocessors.

TABLE II. Processor Specifications

Processor	***Intel Core i7-2600***
# of Cores	4
# of Threads	8
Clock Speed	3.4 GHz
L1 Data	32 KB
L2 (Unified)	256 KB
Third Level (LLC)	8 MB
RAM	16GB

The application used for obtaining metrics was Word Count of Phoenix. This application counts the frequency of occurrence for each word in a set of files. Two datasets were used for benchmarks, one consisting of 50MB and one of 100MB. These datasets were used because of their great sizes that lead the cores to seek shared memory resources.

To analyze the behavior of Phoenix MapReduce, Callgrind was used, a tool provided by Valgrind [15], an instrumentation framework for building dynamic analysis tools. Callgrind is a call-graph generating cache profiler that records the call history among functions in a program's run as a call-graph. Optionally, cache simulation and/or branch prediction can produce further information about the runtime behavior of an application.

Two configurations were used for the scope of this work.

- For the first configuration, Phoenix was set to use 4 cores in order to run Word Count, with a 50MB text file input.
- For the second configuration, Phoenix was set to use 8 cores in order to run Word Count, with input a 100MB text file input.

The profiling results for these configurations that will be used here are the LLC references, which are illustrated in Table III, for each thread of the system. This metric was used because of the high latency, and as a result the bottleneck encountered in LLC memory. Ten runs for each configuration were profiled, and the results from a random and the average run were used in order to evaluate any variation in the performance of the system, caused by scheduling, due to the fact that for all runs, no parameter of Phoenix was altered.

This work is focused in the analysis of the LLC references of each thread, because this is the main bottleneck in a multicore processor. Obviously, L1 and L2 cache are very important for the performance of a processor, however, here, the subject is the conflict of the threads to obtain common resources. As a result, the behavior of these caches is utilized so as to have them as busy as possible along with the profiling procedure, where the exact applications were running in order to minimize the usage of these memories by MapReduce and maximize the MapReduce interference with LLC

It must be noticed here, that Phoenix is really faster than the times presented here. It takes from 3 to 4 seconds to carry out any of the configurations. However, running Phoenix under the profiling of Callgrind, decelerate the operations significantly. As a result, to evaluate the performance of the system, the normalized time elapsed will be used, as shown in Table IV. Also the LLC references of every thread are normalized to the higher value of the LLC references of one thread.

TABLE III. LLC References From Profiling Results

#Thread	Random run for first configuration	Average run for first configuration	Random run for second configuration	Average run for second configuration
1	406.464.194	404.382.927	14.668.989	10.668.023
2	309.057.180	215.988.481	10.309.051	10.624.537
3	38.092.911	26.600.327	18.644.130	24.341.572
4	6.544.671	913.410	59.752.921	28.841.226
5	-	-	14.372.546	40.294.950
6	-	-	121.829.852	59.416.156
7	-	-	39.084.467	85.406.044
8	-	-	335.067.602	394.640.816
Time elapsed (sec)	294	273	534	506

In order to reproduce these behaviors with the theoretical model, two configurations were also created.

- Firstly, to simulate the conflict among 4 cores, a grid of 2×2 players is set. The Moore neighborhood is selected and the boundary conditions are adiabatic. As a result, each player has 3 neighbors - all the other participants.

- Secondly, to simulate the conflict among 8 cores, a grid of 3×3 players is set, but the central player is set to busy mode, and as a result it does not interact with the others. The Moore neighborhood is selected and the boundary conditions are periodic. Consequently every player has 7 neighbors - all the other participants.

The strategies of the players used in the CA grid, were chosen based on the data derived from Table IV. The normalized to the highest LLC references amount, results corresponds to the possibility of each player to defect. As a result the player representing the most "greedy" core of the processor, that has a normalized LLC references value of 1.00, will adopt the defective strategy. The normalized to the highest amount of LLC references, results with values under 0.08 are considered of low significance for the simulation, as they refer to LLC rarely, and so the threads acquiring these results are simulated as players following cooperative strategy. For the other threads, their metric results were rounded up and used as the possibility of the random strategy to defect. The rounded amounts were used in order to have a better understanding and comparison of the type of strategy used. Note here, that as Phoenix runtime assigns tasks dynamically [7] to threads and some tasks may need more resources than others, using the random strategy of the proposed model, is a safe choice.

Consequently, taking into account Table IV, for the random run of the first configuration, there is a defective, a cooperative and two random (one with 80% possibility to defect and the other with 10%) players. For the average run there are two cooperative, one defective and one random (with 50% possibility to defect) player. Furthermore, for the random run of the second configuration, there is one defective, four cooperative and three random (with 40, 20 and 10% possibility to defect) players. Finally, for the average run of the second configuration, there is one defective, four cooperative and three random (one with 20% and two with 10% possibility to defect) players.

TABLE IV. NORMALIZED PROFILING RESULTS

#Thread	Random run for first configuration	Average run for first configuration	Random run for second configuration	Average run for second configuration
1	1	1	0,04	0,03
2	0,76	0,53	0,03	0,03
3	0,09	0,06	0,06	0,06
4	0,02	0,002	0,18	0,07
5	-	-	0,04	0,10
6	-	-	0,36	0,15
7	-	-	0,12	0,22
8	-	-	1	1
Normalized time elapsed	1	0.93	1	0.95

The results of these configurations acquired with the model after 200 rounds are shown in Figs. 1 and 2 per thread and in total in Table V. The score, the community of players collected, as mentioned in Section II, corresponds to the LLC references attempted by each core in a time interval. Consequently, a higher total score means a higher performance for the system, which can be translated as less execution time. The reason for presenting the normalized score and the inverted normalized score in Table V is to be able to compare the results acquired by the model with the ones obtained from profiling a real system.

Note that in Figs. 1 and 2, the *x* axis describes the strategy of every player, participating in the game. The first line (blue) describes the random run and the second line (red) the average run of the application in the same configuration. Furthermore, the *y* axis depicts the LLC references succeeded by every player at the end of 200 rounds.

Furthermore, in order to increase the performance of the system, the GA strategy was adopted by some players of the community. The better performance of the system is expected due to the fact that while there is the same computation load, it is assigned to more threads than the previous configurations. As a result, the load is evenly assigned and the system executes the application faster and more efficiently. The simulation configurations run for 200 rounds with the strategies described above, with some strategies altered to GA.

TABLE V. RESULTS ACQUIRED WITH THE MODEL

	Random run for first configuration	Average run for first configuration	Random run for second configuration	Average run for second configuration
Normalized time elapsed	1	0.93	1	0.95
Total score of the community	5662	6094	30884	31468
Normalized score	1	1.076	1	1.02
Inverted normalized score	1	0.929	1	0.98

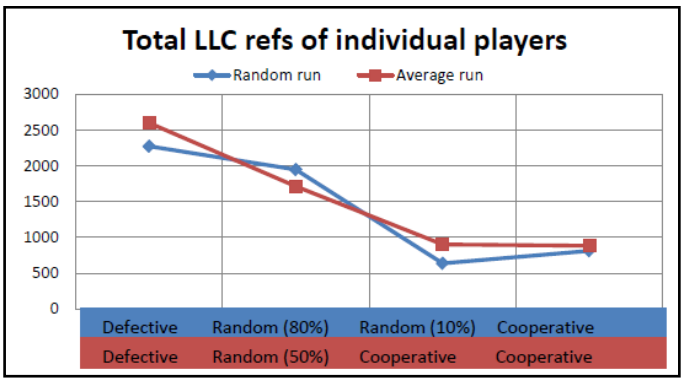

Figure 1. Results per core from simulation with 4 cores after 200 rounds. The *x* axis represents the strategy followed by an individual player/core throughout the Random and the Average run.

For the random run of the first configuration of the model, simulating the 4 cores instance, the static strategies (defective

and cooperative) are changed into GA strategies. On the other hand the random strategies remain, keeping the possibilities used previously. This way the whole system becomes dynamic and also uses an adaptive method to spread the load. The results after 200 rounds for the random and the optimized run are illustrated in Fig. 3.

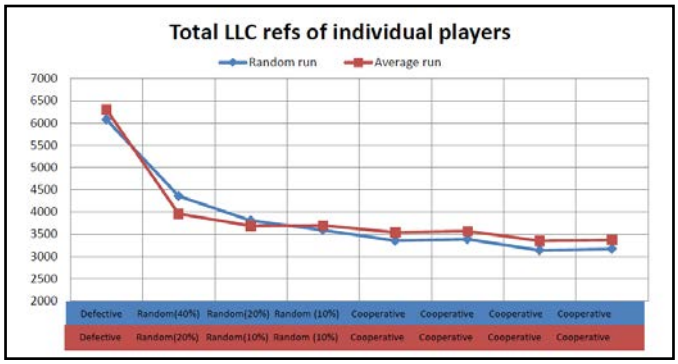

Figure 2. Results per core from simulation with 8 cores after 200 rounds. The *x* axis represents the strategy followed by an individual player/core throughout the Random and the Average run.

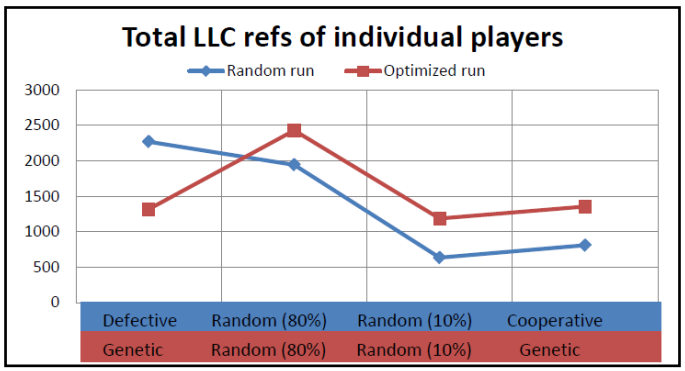

Figure 3. Results per core from simulation with 4 cores after 200 rounds. The *x* axis represents the strategy followed by an individual player/core throughout the Random and the Optimized run.

Furthermore, for the random run of the second configuration of the model, simulating the 8 cores instance, three players were chosen to change their strategies (two cooperative and the defective). Here, only three static strategies change, because the computation load of the defective player can be sufficiently divided to three players. Splitting the load of one player to a bigger number of players, would have the opposite results. The results after 200 rounds for the random and the optimized run are illustrated in Fig. 4.

Finally, the comparison of the overall results of the community for the random and the optimized runs, are presented in Table VI. There is an increase in the performance of the system, 10% for the first configuration and 2.3% for the second configuration.

It must be pointed out, that for the optimization of the system in both configurations, static strategies were chosen to change into GA ones, because these strategies cannot reply to the changing dynamics of random players – or in a real system with the altering of memory needs by some cores. Furthermore, many strategies were kept the same, in order to have a comparison with the applications used in the profiling of the real system.

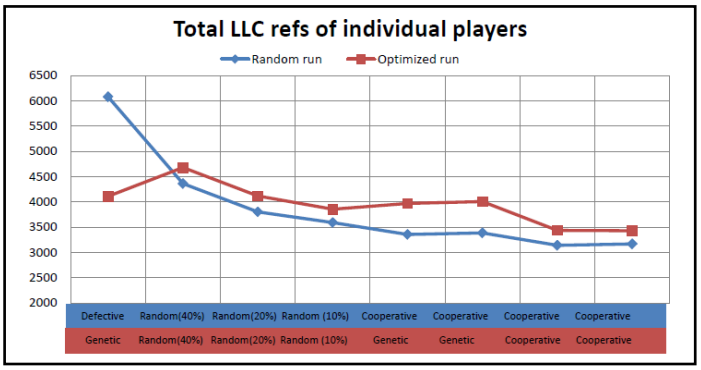

Figure 4. Results per core from simulation with 8 cores after 200 rounds. The *x* axis represents the strategy followed by an individual player/core throughout the Random and the Optimized run.

TABLE VI. RESULTS ACQUIRED WITH THE MODEL

	Random run for first configuration	Optimized run for first configuration	Random run for second configuration	Optimized run for second configuration
Total score of the community	5662	6286	30884	31606
Normalized score	1	1.11	1	1.023
Inverted normalized score	1	0.9	1	0.977

V. CONCLUSIONS AND DISCUSSION

In this paper, the memory resources of a multicore processor system, in specific the on-chip memory per processor redistribution, so as to meet higher performance based on the basic concepts of game theory with the help of adaptive computational tools like CAs, GAs and the iterated spatial prisoner's dilemma game was examined. The proposed model originally intrigued by a classic model of the dynamics of cooperation and noncooperation, namely the iterated spatial prisoner's dilemma, was depictured in a CA lattice and in order to take advantage of "rational", namely genetic, strategies used the above game as the presented CA evolution rule.

In the view of the foregoing, software simulations were established, in order to reproduce the results of the metrics obtained by profiling tools running on a multicore shared-memory system. As presented by the simulation results, it can be safely admitted that the theoretical results can approach the results of a real multicore system. However, the deflection noticed to the theoretical results compared to the real system metrics, can be explained by the simplicity of the model and the use of specific metrics of the real system, namely, only LLC references. Using metrics of more shared resources, such as the communication bus, and a more complicated model, will, assumingly, lead to more accurate simulations.

Moreover, the usage of adaptive strategies revealed the potential enhancement of the performance of the system. Nonetheless, the parameters that initialize the theoretical model enable the simulation of the vast majority of multicore systems. The most significant of these parameters are the number of players, the type of neighborhood, the local rule strategy and the payoff table.

Nevertheless, as future work, some more technical details regarding the on-chip memory usage in multicore processors could be also taken into account in order to alter the payoff table to obtain results closer to the real system metrics. Some strategies could be also further differed, after the mapping of needs, for utilization of various resources, of often used applications. Furthermore, profiling different configurations of Phoenix MapReduce can provide more case studies to compare with results from the model. These configurations may be another amount of cores participating in the operation, another amount of Map or Reduce workers and different amount of tasks. Finally, the implementation of rational, more adaptive strategies with the help of evolutionary or swarm intelligence techniques in the proposed model, especially applied to the selection of the under study CA rules could also lead to a greater performance in memory management of the system.

ACKNOWLEDGMENT

The research project is implemented within the framework of the Action "Supporting Postdoctoral Researchers" of the Operational Program "Education and Lifelong Learning" (Action's Beneficiary: General Secretariat for Research and Technology), and is co-financed by the European Social Fund (ESF) and the Greek State.

REFERENCES

[1] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, D. Burger, "Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures," In Proceedings of the 27th Annual International Symposium on Computer Architecture, June 2000, pp. 248–259.

[2] D. Abts, N. Jerger, J. Kim, D. Gibson, M. Lipasti, "Achieving Predictable Performance through Better Memory Controller in Many-Core CMPs," In Proceedings of ISCA, 2009, pp. 451-461.

[3] M. Awasthi, D. W. Nellans, K. Sudan, R. Balasubramonian, A. Davis, "Handling the problems and opportunities posed by multiple on-chip memory controllers." In Proc. 19th international conference on PACT'10, 2010, pp.319-330.

[4] B. Stengel, Game Theory Basics, London School of Economics. 2008

[5] O. Durán, R. Mulet, "Evolutionary Prisoner's Dilemma in Random Graphs," Physica D,vol. 208(3-4), 2005, pp. 257-265.

[6] A. Herdrich, R. Illikkal, R. Iyer, D. Newell, V. Chadha, J. Moses, "Rate-based QoS techniques for cache/memory in CMP platforms." In Proc. 23rd international conference on Supercomputing (ICS'09), 2009, pp. 479-488

[7] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, C. Kozyrakis. "Evaluating MapReduce for Multi-core and Multiprocessor Systems," In Proc. 13th Intl. Symposium on High-Performance Computer Architecture (HPCA), February 2007, pp. 13-24.

[8] R. Myerson, Game Theory: Analysis of Conflict, Harvard University Press. 1997

[9] Y. Kim, D. Han, O. Mutlu, M. Harchol-Balter, "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers," In Proc. 16th International Symposium on High Performance Computer Architecture, 2010, pp. 1-12.

[10] J. Mars, N. Vachharajani, M. L. Soffa, R. Hundt, "Contention aware execution: Online contention detection and response." In Proc. 8th annual IEEE/ACM international symposium on Code generation and optimizatio}, CGO'10, pp. 257-265

[11] J. Vukov,G. Szabó, A. Szolnoki, "Evolutionary prisoner's dilemma game on Newman-Watts networks," Physical Review E, vol. 77, 026109, 2008.

[12] M.I. Tsompanas, G.Ch; Sirakoulis, I. Karafyllidis, "Modeling memory resources distribution on multicore processors using games on cellular automata lattices," in Proceedings of IPDPS, 2010, pp. 1-8.

[13] D. Vantrease, R. Schreiber, M. Monchiero, M. McLaren, N. Jouppi, M. Fiorentino, A. Davis, N. Binkert, R. Beausoleil, J.H. Ahn, "Corona: System Implications of Emerging Nanophotonic Technology." In Proceedings of ISCA, 2008, pp. 153-164.

[14] D. Wentzlaff, P. Griffin, H. Hoffmann, Bao; Liewei B. Edwards, C. Ramey, M. Mattina, M. Chyi-Chang, J.F. Brown, A. Agarwal, "On-Chip Interconnection Architecture of the Tile Processor," In IEEE Micro, vol. 27(5), 2007, pp. 15-31

[15] J. Weidendorfer, M. Kowarschik, C. Trinitis, "A Tool Suite for Simulation Based Analysis of Memory Access Behavior," Proceedings of the 4th International Conference on Computational Science (ICCS 2004), Krakow, Poland, June 2004.

[16] G. Ch. Sirakoulis, I. Karafyllidis, A. Thanailakis, and V. Mardiris, "A methodology for VLSI implementation of Cellular Automata algorithms using VHDL," Advances in Engineering Software, vol. 32, no. 3, 2001, pp. 189-202.

[17] G. Ch. Sirakoulis, I. Karafyllidis, and A. Thanailakis, "A CAD system for the construction and VLSI implementation of Cellular Automata algorithms using VHDL," Microprocessors and Microsystems, vol. 27, no. 8, 2003, pp. 381-396.

[18] B. Chopard, and M. Droz. Cellular Automata Modelling of Physical systems. Cambridge: Cambridge University Press, 1998.

[19] V. Mardiris, G. Ch. Sirakoulis, Ch. Mizas, I. Karafyllidis, and A. Thanailakis, "A CAD system for modeling and Simulation of Computer Networks using Cellular Automata," IEEE Transactions on Systems, Man and Cybernetics – Part C, vol. 38, no. 2, 2008, pp. 253-264.

[20] G. Ch. Sirakoulis, "A TCAD system for VLSI implementation of the CVD process using VHDL," Integration, the VLSI Journal, vol. 37, no. 1, 2004, pp. 63-81.

[21] J. von Neumann, Theory of Self-Reproducing Automata. University of Illinois, Urbana, IL, 1966.

[22] G.Ch. Sirakoulis, and I. Karafyllidis, "Cooperation in a Power-Aware Embedded System Changing Environment: Public Goods Games with Variable Multiplication Factors," IEEE Transactions on Systems, Man, and Cybernetics–Part A: Systems and Humans, vol. 42, no. 3, 2012, pp. 596-603.

[23] J. Neel, A.B. Mackenzie, R. Menon, L.A. Dasilva, J.E.Hicks, J.H. Reed, R.P. Gilles, "Using game theory to analyze wireless ad hoc networks," IEEE Communications Surveys and Tutorials, vol. 7, no. 4, 2005, pp. 46-56.

[24] A. MacKenzie, S. Wicker, "Game theory and the design of self-configuring, adaptive wireless networks," IEEE Communications Magazine, vol. 39, no. 11, 2001, pp. 126-131.

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

[26] D. Puschini, F. Clermidy, P. Benoit, G. Sassatelli, L. Torres, "Temperature-Aware Distributed Run-Time Optimization on MP-SoC Using Game Theory," Symposium on VLSI, 2008. ISVLSI '08. IEEE Computer Society Annual, 2008, pp.375-380.

[27] A. Piwonska, F. Seredynski, "A Genetic Algorithm with a Penalty Function in Selective Travelling Salesman Problem on a Road Network," 2011 IEEE International Parallel & Distributed Processing Symposium, 2011, pp.381-387..

[28] S. H. Heap, Y. Varoufakis, Game Theory: A Critical Text. Routledge, New York, 2004

[29] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs. Springer, 1998