Evaluating Conflicts Impact over Shared Last-Level Cache Using Public Goods Game on Cellular Automata.

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*— Regardless the fact that micro-architecture design has reached a very high level of complexity to overcome the restrains of Moore's Law, memory systems remain a bottleneck for processing systems. Taking that into account, this paper presents a model that simulates the conflict between cores for the use of shared resources, i.e. cache memory, on a multi-core processor and illustrates the impact of that conflict on the global performance of the system, using an example of Game Theory, namely Public Goods Game. In this context, Cellular Automata are the computational tool selected, in order to encapsulate the local dynamics realized on a multi-core system. Moreover, in order to validate the applicability of the model, the behavior of multicore applications running on a regular multi-core system, were profiled and compared with the results obtained from the model. This comparison illustrated that the performance of system simulated by the model is in good agreement with the profiling results of the real system configuration.**

Keywords-component; cellular automata; game theory; multi-core systems; memory resources allocation

I. INTRODUCTION

Nowadays multi-core processors have a key role in computer industry, being a mainstream in the computer market. Also, they steer computing performance in higher standards. However, the continuous need for more power and, at the same time, energy efficient systems, leads to more complicated solutions, such as processors with higher amount of cores on one chip [1], [2] and more complex architectural and memory structures. On the other hand, the development of operating systems (OS) has not produced a rather sufficient method of leveraging the new hardware. Moreover, memory systems are still a huge bottleneck compared to the high clock frequencies of processors, taking into consideration the fact that for die area economy, some memory systems are shared to many cores of the processor. Some innovative architectural methods, like prefetching and out-of-order execution, that aim to decrease the latency introduced by memory do not have the same effects on every case [3]. The gap between processor and memory performance, that was early foreseen [4], will continue to grow.

In order to deal with this problem, the research community produced a wealth of studies, on how applications can be co-scheduled in different cores of a processor so as to use efficiently the common resources and more specifically the, die-area expensive, on-chip memory. Some methods are based on software [3, 5, 6], while others on hardware [7, 8]. Moreover, some rely on both software and hardware [9, 10]. The vast majority of that research is confirming the fact that two applications, thus the cores executing them, can "cooperate" over common resources, or "contend" for the selfish occupation of it. The term "cooperation" is leading to game theory, which is defined as the analysis of mathematical prototypes of collaboration and antagonism between intelligent rational decision-makers [11].

Furthermore, one of the most fitting, here, example of game theory is the public goods game (PGG) [13], because there are cores that compete for public resources, which are affected by the decisions made. Moreover, to be consistent with the technology advances, Cellular Automata (CA) concept can be considered as a wise choice, taking into account that processors are designed with greater amount of cores, which are found in a more mesh or grid-like regular structure inside the processor and the distribution of the shared resources is becoming more complex [1], [2]. For instance Intel's Single-Chip Cloud Computer (SCC), which is an experimental multi-core processor, has 48 Pentium (P54C) cores in 24 tiles of two cores each and the tiles are connected by a four by six mesh in the chip. This is an issue that CA can deal with, due to their ability to capture inhomogeneities with their local rule when local interactions appear.

As a result, a model was produced using game theory concepts, and more specifically PGG, on CA lattice, to simulate the impact of the conflict of cores for shared Last Level Cache (LLC) resources. The results of the model compared to the performance of the applications on a real system are found in good agreement, particularly taking into account the simplicity of the model and the, relatively, hardware agnostic method used.

The remainder of the paper is organized as follows. In Section II some related works and the motivation for the proposed model are discussed. In Section III, to make some definitions clear, a theoretical background of the presented model preliminaries is given, while in Section IV, the proposed model is described in full detail. Finally, in Section V the real system profiling methodology is presented and its

results are compared with the results obtained from the proposed model. Conclusions and some future work are discussed in Section VI.

II. Related Work And Motivation

A great amount of studies deal with the contention and cooperation over shared resources on a multi-core system. The majority of them review the shared on-chip memory impact on the overall performance of the system. As mentioned before, many researchers rely on software methods, others on hardware, while others combine the advantages gained from both methods.

In specific, Zhao et al. presented a hybrid last-level cache design and examined the effects it had, in terms of miss and hit rate, for several significant server applications and multi-programmed workloads [7]. In more detail, each cache slice was separated into a private and a shared slice and a directory cache was developed in order to point to the remote private cache location. The estimated effect of multi-programmed and multi-threaded workloads, processed under the proposed architecture is the improvement of local hit rate up to 90% while keeping the miss rate close to that of a shared cache.

Ebrahimi et al. realizes the need of a unified method that will treat fairly the entire shared memory system and suggested an inexpensive architectural technique, named Fairness via Source Throttling (FST) that permits the execution of software fairness policies, by enabling fair sharing of the entire memory system [8]. The mechanism of FST is gathering hardware metrics denoting the execution delay of different applications and, depending on that information, changes the combined memory requests of sources to scale these delays.

On the other hand, from the software point of view several impressive results are presented. CASC [3], a cache-aware operating system scheduling algorithm for multithreaded chip multiprocessors is proposed. CASC is based on scheduling together threads that when combined, perform low L2 cache miss rate and characterizing urgent, threads that demand a little proportion of the L2 cache. CASC results on a reduction of L2 miss rate that ranges from 15% to 46% and throughput enhancement of the processor from 28% to 50%, by approximating the miss rate of L2 at runtime, without an important performance overhead.

Moreover, Zhuravlev and his colleagues, investigated how and to what extent rivalry for shared resources can be reduced by implementing thread scheduling [5]. The most challenging aspect of this work is assumed to be the classification scheme for the threads that will decide the way they interfere with others, while defecting for shared resources. One of the outcomes of that work is that using information of the miss rates as the factor of the classification scheme, makes possible the diminishing of competition for shared resources by changing some scheduling options. Moreover, a scheduling method that is sensitive to contention, will enhance the global efficiency of the system. Finally, other aspects like memory controller, memory bus and prefetching hardware conflicts, are valuated as very important to the system's performance, however in order to mitigate these conflicts, the reduction of the total number of cache misses is considered mandatory. As a result, two scheduling algorithms, namely Distributed Intensity and Distributed Intensity Online, were finally proposed.

Tang et al. introduced a detailed investigation of datacenter applications interacting on the shared resources constituting a memory system [6]. An improvement of 25% for web search was realized, as a result of just efficient allocation of threads through cores. Consequently, optimizing the thread-to-core allocation of applications in datacenters is of paramount importance. Furthermore, important features of applications were studied, as these features affect the optimal thread-to-core allocation method.

Nonetheless, researchers proved that methods that take advantage of both, software and hardware optimizations, present really good results. Following this principle, Ghosh et al. suggested hardware support for reducing the interference produced from L2 cache conflicts [9]. Three resource allocation algorithms were suggested, aiming to mitigate the interference. Jaleel et al. designed a method using both software and hardware for controlling the shared cache, named Cache Replacement and Utility-aware Scheduling (CRUISE) [10]. This method is aware of the LLC replacement policy applied and application cache utility information to dictate how the applications will be divided in the best way possible. Also, it was proved that the weight of software to intelligent scheduling was minimized from smart cache replacement; however the demand for determining the best application co-schedules was not completely eradicated. Furthermore, a Runtime Isolated Cache Estimator (RICE) was presented, a procedure using hardware to dynamically resolve the performance of an application on an isolated LLC while at the same time it uses the shared LLC. Finally, a categorization of applications by cache utility was proposed:

- "Core Cache Fitting" Applications,
- "LLC Thrashing" Applications,
- "LLC Fitting" Applications and
- "LLC Friendly" Applications.

In an analogous mater, from a game-theoretic point of view, the applications can be categorized as fully defective, merely defective, and merely cooperative. This characterization will be based on their need for shared cache resources.

III. Theoretical Background

In order to apprehend the dynamics between cooperation and competition, a mathematical tool is needed. Game theory is used vastly in occasions involving rational decision-makers that their decisions interfere with each other gains. A commonly used paradigm of game theory is the PGG [13], which presents the interactions of individuals constituting a group. For instance, some individuals are awarded with an equal amount of money. Afterwards, the individuals are facing a challenge; to invest in total or a part of the initial amount awarded into a common pot, being aware that the common pot raised, will be multiplied and divided equally to all of them, regardless the contribution each one made. In the case that everyone invests the entire initial amount, everyone

will get a greater amount of the money invested. Still, each one is tempted to "free-ride" on the investments made by other members of the group, since this way there is no risk for his initial capital. Assuming that all players follow this "rational" strategy, the initial capital will remain static [12].

From a theoretical point of view, players participate in a public good game in groups of n players. The game is elapsed t rounds. On every round of the game, a player i obtains an award w and faces the dilemma on how to divide it. He must choose between an investment c_i, to a public good, the common pot, and private utilization, $w - c_i$. The total amount invested by the n members of a society is multiplied by β, $\beta < n$, and equally distributed to the n members [13]. Denoting $m = \beta / n$, $m < 1$, the payoff of player i at each round as a function of the contribution to the public good is illustrated by (1). Finally, at the beginning of the next round, the obtained award w for a player i, will be equal to the payoff π_i gained by the player at the previous round.

$$\pi_i = w - c_i + m \sum_{i=1}^{n} c_i \quad (1)$$

In PGG, the public good or environment is depicted by the multiplication factor β, that in the case of remaining constant during the game, the public good cannot be totally consumed by players that adopt "wrong" strategies. The amounts that are chosen by the players for investment have an impact on the production of the common good that will be equally divided among them [15].

Common sense dictates that in a society that is donating itself with a public good, every individual constituting it will be highly tempted to become a free rider, meaning to give in little or nothing at all, with consequences to the welfare of the community and at the same time receiving the rewards everyone else receives. The fact that the phenomenon of free riders will cause the community to provide its members with less rewards, is also predicted by economic theory [14]. Moreover, supremacy of asocial, defecting strategies is prognosticated by traditional and evolutionary game theory. On the other hand, a permanent and strong willingness to cooperate in societies is significant [12]. It becomes impressive to differentiate from theoretical prognostications, granted the significant obstacles to establish and maintain cooperative behavior in large groups [16].

However, the progress in theory and experiments has demonstrated some methods that are capable of encouraging cooperation. Many modifications of the classic PGG have been proposed, including spatial PGG [17], [18], and PGG in which the players are separated in groups [19]. In spatially extended systems cooperators can have great advantages when they form clusters that reduce exploitation through defectors [12]. As a result the use of Cellular Automata (CA) comes into hand, taking advantage of their ability to successfully depict local interactions and incorporate inhomogeneities in their local rule.

CA are models of physical systems, where space and time are discrete and interactions are local [22]. In this section a formal definition of a CA will be presented [23]. In general, a CA requires:

1. A regular lattice of cells covering a portion of a d–dimensional space;
2. A set $\mathbf{C}(\vec{r}, t) = \{C_1(\vec{r}, t), C_2(\vec{r}, t), ..., C_m(\vec{r}, t)\}$ of variables attached to each site $\vec{r}$ of the lattice, giving the local state of each cell at the specific time value t;
3. A rule $\boldsymbol{R}=\{R_1, R_2, \ldots, R_m\}$ which specifies the time evolution of the states $\boldsymbol{C}(\vec{r}, t)$ in the following way: $C_j(\vec{r}, t+1) = R_j(\boldsymbol{C}(\vec{r}, t), ..., \boldsymbol{C}(\vec{r} + \vec{\delta}_q, t))$, where $\vec{r} + \vec{\delta}_k$ designate the cells which belong to a given neighbourhood of cell $\vec{r}$.

In the above definition, the rule $\boldsymbol{R}$ is identical for all sites and it is applied simultaneously to each of them, leading to synchronous dynamics. However, spatial (or even temporal) inhomogeneities can be introduced. Furthermore, in the above definition, the new state of a particular cell r at time t+1 is only a function of the previous state of the specific cell and of the cells which belong to its designated neighbourhood. The neighbourhood of cell $\vec{r}$ is the spatial region in which a cell needs to search in its vicinity. For 2–d CA, two types of neighbourhood are usually considered: namely, von Neumann neighbourhood, which consists of a central cell (the one which is to be updated) and its four geographical neighbours north, west, south and east and Moore neighbourhood which contains, in addition, second nearest neighbours northeast, northwest, southeast and southwest, i.e. a total of nine cells, whereas the von Neumann neighbourhood comprises of only five cells. CA have sufficient expressive dynamics to represent phenomena of arbitrary complexity [24] 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 [25]. 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. Moreover, the implementation of CA rules in VLSI circuits or FPGA logic will enhance their inherent parallel nature [26], [27]. Furthermore, they can easily handle complicated boundary and initial conditions, inhomogeneities and anisotropies.

IV. The Proposed Model

As mentioned in Section II, the subject of many studies is the conflict and the cooperation for the shared system resources, between applications, and thus the cores that are assigned to complete them, on multi-core architectures. Consequently, a model is proposed here, to simulate that conflict between cores and to estimate its impact over the global system performance. The rules that apply on that situation are considered to be in accordance with the PGG described in Section III. It must be noted here, that the choice of representing cores as the players in a PGG, instead of applications or processes, is due to the fact that the model

is hardware oriented and the profiling of the real system, in order to have a comparison, is made per core. Also, a simulation of the behavior of applications would have been much more complicated, because of some mechanisms used in the up-to-date processors, like CPU migrations.

The individuals of the proposed model are regarded as the cores of a multi-core system and it is assumed that they are identical, thus they are represented as PGG players placed in CA cells in a square grid. The cores, in a real system, will be assigned with some tasks and will compete for the use of the shared memory. As a result, the reward of every CA cell will simulate the accessibility to the LLC of the system for approximately the same period of time for a single core. That means that the public good will be assumed to be the total amount of references to the available LLC for a time period. Moreover, the investment of a player placed in a CA cell in every round will correspond to the amount of LLC resources that the core does not need and can be used from other cores. As the amount of LLC accessibility by a core is modeled as the payoff of every player, the available common good through time depends on the payoffs of the players.

The players of the proposed model are placed in a square CA grid. Each player interacts with his neighbors, as they constitute a community of a PGG. Furthermore, the type of the neighborhood and the boundary conditions can be altered in order to depict different shared cache systems. To simulate an architecture that is consistent with the up-to-date, easy to access, commercial multi-core processors and, more specifically, the processor (Intel Core i7 2600) used as described in Section V, the following choices were made. The neighborhood type selected was Moore, the boundary conditions are periodic and the grid is 3×3, in order to simulate a system of 9 players competing over a single shared LLC. By setting one player to busy mode, the resulting architecture is a community of 8 cores with one shared LLC. However, the model is not restricted by these options. Other architectures [1], [2] can be simulated by using a larger grid and neighborhood's radius, however as these processors are not easy to access, this will be the subject for a future study.

Another parameter of the model is the time steps, namely the game rounds, here empirically chosen equal to 100. Furthermore, the multiplication factor β is set to 6, a value lower than the amount of cores, n=8, in order to keep the social dilemma. The multiplication factor can be altered to a constant or dependant by time value to simulate different system circumstances. The gain of every player *(i,j)* on round *t* for the configuration described above is given by Equation (2).

$$Gain^{t}_{(i,j)} = \frac{\beta}{n}(Investment_{(i-1,j)} + Investment_{(i,j-1)} + Investment_{(i+1,j)} + Investment_{(i,j+1)} + Investment_{(i-1,j-1)} + Investment_{(i-1,j+1)} + Investment_{(i+1,j-1)} + Investment_{(i+1,j+1)} + Investment_{(i,j)}) \quad (2)$$

Moreover, the payoff of every player (*i,j)* on round *t*, is given by Equation (3).

$$Payoff^{t}_{(i,j)} = Payoff^{t-1}_{(i,j)} - Investment_{(i,j)} + Gain^{t}_{(i,j)} \quad (3)$$

Furthermore, the amount of the investment of every player is determined by the strategy it adopts. Players with investment value 0 are defectors and represent cores that need an excessive amount of LLC. Also, players choosing investment value 1, namely cooperators, represent cores that need a very small amount of LLC and do not interfere significantly with the others' needs. Moreover, every player can choose intermediate values to invest, simulating the proportional need of LLC.

Finally, the total payoff of a core at the end of the last round will be the sum of the rewards obtained for all previous rounds. As the payoff of each player on one round represents the ability to access the same amount of LLC of the system for a period of time, the total payoff of the group will represent the available utilization of the LLC that is corresponding to the performance of the system.

V. METRICS

In order to evaluate the results of the model, metrics on a real system have been obtained. For the acquisition of these metrics the system described in Table I was used. Intel Core i7 2600 processor chip was used, which includes four CPU cores and on-chip cache memory on a 45nm die. Also, hyperthreading technology makes the software run as if there were eight processing units. Each of the four cores has a 32KB instruction and a 32KB data Level 1 cache, and a 256KB of Level 2 cache. However, the four physical cores, and the eight virtual, share an inclusive 8MB Level 3 cache, here denoted LLC.

The multithread applications selected to run on the aforementioned system, are three of Phoenix MapReduce [20] runtime implementations, namely Word Count, Histogram and String Match. 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 [20].

Word Count counts the frequency of occurrence for each word in a file. String Match processes two files: the "encrypt" file contains a set of encrypted words and a "keys" file contains a list of non-encrypted words. The goal is to encrypt the words in the "keys" file to determine which words were originally encrypted to generate the "encrypt file". Histogram analyzes a given bitmap image to compute the frequency of occurrence of a value in the 0-255 range for the RGB components of the pixels. Furthermore, the input files for every example used were the largest available from the developers of Phoenix [20], in order to simulate a situation that need a lot of resources. For String Match a 500MB file was the "keys" file and the "encrypt" file was

implemented in the user defined code, for Histogram a 1.4GB file and for Word Count a 100MB file were used. As a result, Map functions that are executed in parallel on non-overlapping portions of the input data, will require a lot of memory resources which are not considering the same process space.

In order to obtain the profiling information from Phoenix MapReduce examples, Callgrind was used, a tool provided by Valgrind [21], 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. Moreover a choice of cache simulation and branch prediction can be made, that generates more information about the runtime behavior of an application. Cache simulation used in Callgrind is based on that of Cachegrind, another tool provided by Valgrind.

TABLE I. SYSTEM'S SPECIFICATIONS

Processor	*Intel Core i7-2600*
# of Threads	8
Clock Speed	3.4 GHz
L1 Data	32 KB
LLC	8 MB
RAM	16GB
OS	Ubuntu 12.04 (Linux Kernel 3.2)

Word Count was run ten (10) times and was profiled each time, in order to have a complete picture of its behavior under different scheduling schemes that were introduced during every run by the default scheduler of the operating system. Also, the same procedure was followed for Histogram and String Match. Phoenix parameters were not altered for any run and it used all 8 cores of the system.

After ten (10) runs of Histogram with a 1.4GB file as input, the results from the profiling showed a little variation on execution time from a 3% faster to a 5% slower from the average. Also, the total LLC references experienced do not exceed 2.2% of data references, the LLC references per Kilo Instructions (RPKI) per core are pretty consistent, from 12 to 14 (16 to an extreme situation) and as a result this application is little affected by the different scheduling schemes adopted by the operating system. In Fig. 1 the execution time for 10 runs of Histogram are depicted.

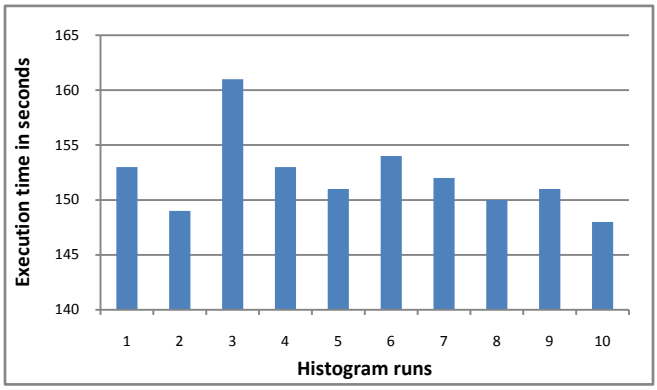

Figure 1. Histogram results.

After ten (10) runs of String Match with a 500MB file as input, the results from the profiling demonstrated a negligible variation on execution time from 0.8% faster to 1.38% slower from the average. Furthermore, the total LLC references experienced do not exceed 0.057% of data references, the LLC RPKI per core are from 0.2 to 0.4 (8 to an extreme situation). Due to these facts, this application is, also, slightly affected by the different scheduling schemes adopted by the operating system. In Fig. 2 the execution time for 10 runs of String Match are illustrated.

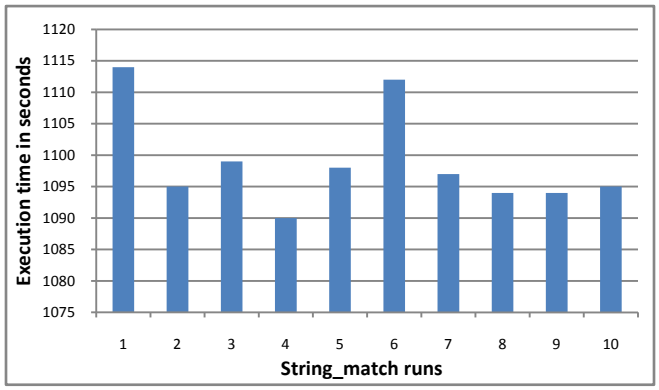

Figure 2. String Match results.

Finally, Word Count was also profiled for ten (10) runs, using a 100MB as an input file. The results showed a significant variation on execution time from 9% faster to 11% slower from the average time. Also, the total LLC references experienced approach 10% of data references and the LLC RPKI per core are extended from 9 to 78. Fig. 3 depicts the execution time for 10 runs of Word Count. Consequently, in an application with a high total LLC references rate and LLC RPKI per core, threads scheduling and allocation of resources will have a significant effect. That can be realized after examining Fig. 4, where the normalized to the average execution time of the best and worst case are depicted.

The simulation of two runs of Word Count was attempted with the proposed model. For these two runs the information used from the real system profiling was the LLC RPKI per core. As illustrated in Table II, these values were normalized to the doubled value of the average LLC RPKI per core for all runs. The average LLC RPKI per core is 37.53, and the doubled value is 75.06. That way, threads that are more data intensive than others or have a high LLC RPKI, are indicated with values near 1. On the other hand, applications that do not need a large amount of LLC have a low LLC RPKI and their normalized values are near 0. Furthermore, cores with a high LLC RPKI, were simulated with defecting players, while cores with a low LLC RPKI, were simulated with cooperating players. As presented in Section IV, cooperators have investment value 1 and defectors 0. Consequently, the values that have been used as investment of every player in the model are presented in Table III and were produced from the subtraction of the normalized values of Table II from 1. Note that there are no

negative values; however these results are rounded to zero in order to represent free-riders. Finally, it must be noticed here, that the placement of the players on the CA grid is not changing the results, as the neighborhood, here, chosen is Moore, and the boundary conditions periodic, because the system simulated consists of 8 cores with one shared LLC.

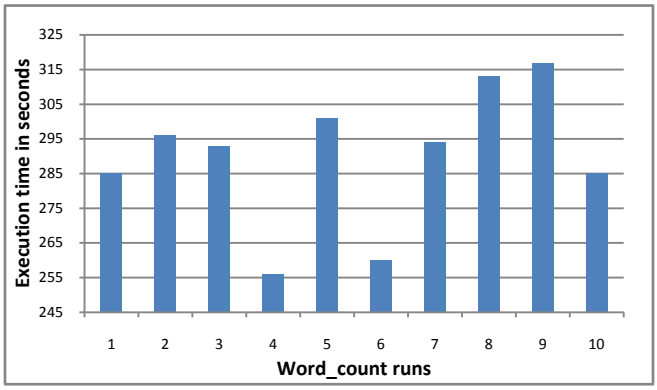

Figure 3. Word Count results.

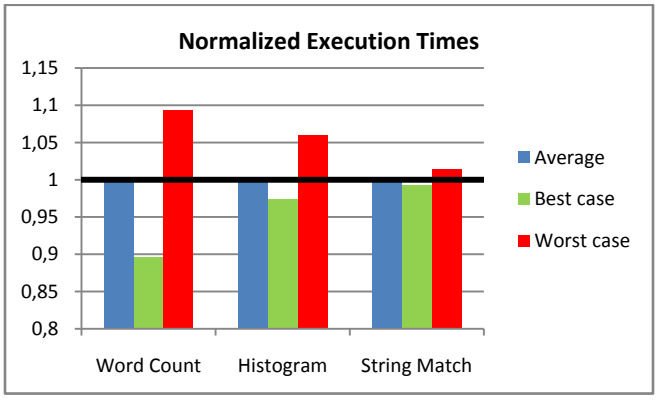

Figure 4. Normalized execution time for the best and worst case for the three application.

TABLE II. WORD COUNT THREADS LLC RPKI

# thread	First run's LLC RPKI	Norm.	Second run's LLC RPKI	Norm.
1	29.02	0.39	79.26	1.06
2	36.70	0.49	15.45	0.21
3	43.76	0.58	76.31	1.02
4	27.30	0.36	67.10	0.89
5	35.74	0.48	12.11	0.16
6	9.72	0.13	9.58	0.13
7	74.83	1.00	78.55	1.05
8	51.47	0.69	11.17	0.15

Using the results from the simulation with the model of the worst and the best case, a comparison between the theoretical model and the real system profiling can be drawn. The execution time for the first run of Word Count on a real system is 260 seconds, while for the second is 317. This means that the first run was 18% faster than the second. Moreover, the total payoff of the players composing the first model is 1963, while for the second 1754. Meaning the first configuration of players' investment policies is 12% more efficient from the second.

TABLE III. WORD COUNT THREADS INVESTMENT VALUES FOR THE MODEL AND RESULTING PAYOFFS

# thread	First model investments	Payoffs	Second model investments	Payoffs
1	0.61	233	0.00	262
2	0.51	243	0.79	184
3	0.42	252	0.00	262
4	0.64	230	0.11	251
5	0.52	242	0.84	179
6	0.87	207	0.87	176
7	0.00	293	0.00	262
8	0.31	263	0.85	178

VI. CONCLUSION AND FUTURE WORK

Drawing inspiration from the technological advances, a theoretical model was proposed that simulates the conflict between cores for the occupation of shared resources, and specifically the on-chip shared memory, on a multi-core processor and depicts its impact on the overall performance of the system, using an example of Game Theory, namely Public Goods Game. Furthermore, CA were used, in order to encapsulate the local dynamics realized on a multi-core system, as manufactures and designers tend to avoid long connections and are keen on local ones.

The configuration used here to present the simulation of an up-to-date, easy to access, commercial multi-core processor, namely Intel Core i7 2600, was a 9 cell CA grid, with one busy cell, and the Moore neighborhood, because the processor has 8 threads and one shared LLC. The results of the model are found in good agreement with the real system profiling results, despite a variation that can be justified by the fact that the only information used is the LLC RPKI per core. However, it is shown that contention can be experienced to other shared resources too, such as the memory controller, memory bus and prefetching hardware [5].

Processor's industry is producing continuously different architectures and complex systems that bear little resemblance to each other. As a result every different on-chip memory hierarchy should be studied separately and simulated under a different configuration of the model. However, the choice of implementing the players on a CA grid, gives the model the opportunity to depict state-of-the-art systems that use more complex on-chip memory architectures. For instance, another processor, i.e. the Dual Socket Intel Clovertown, which is studied in [6], that consists of eight cores and four L2 caches shared by two cores, can be simulated with a CA grid with different type of neighborhood and boundary conditions, in order to make every player interact with only one neighbor. However, the rules of the PPG will remain the same, with the exception of the multiplication factor that will have to represent a different public good.

Also, the alternation of the multiplication factor from a constant value to a time-variable, will give the model the possibility of simulating a different type of environment in which the public good or the LLC availability will be

changing through time. Finally, players can adopt dynamic strategies, in contrary to the static ones adopted here, to simulate real life threads that alternate their needs, throughout the run-time.

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] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, B. Liewei, J. Brown, M. Mattina, M. Chyi-Chang, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook, "TILE64 - Processor: A 64-Core SoC with Mesh Interconnect," IEEE International Solid-State Circuits Conference Digest of Technical Papers, 2008, pp. 88.

[2] J. Howard, S. Dighe,Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, F. Pailet, S. Jain, T. Jacob, S. Yada, S. Marella, P. Salihundam, V. Erraguntla, M. Konow, M. Riepen, G. Droege, J. Lindemann, M. Gries, T. Apel, K. Henriss, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, R. Van der Wijngaart, T. Mattson, "A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS," IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010, pp. 108-109.

[3] A. Fedorova, M. Seltzer, M. Smith, and C. Small, "CASC: A Cache-Aware Scheduler For Multithreaded Chip Multiprocessors" Sun Labs Technical Report, March, 2005.

http://labs.oracle.com/projects/scalable/pubs/CASC.pdf

[4] W. Wulf, and S. McKee, "Hitting the Memory Wall: Implications Of the Obvious," ACM SIGARCH *Computer Architecture News*, vol. 23, issue 1, 1995, pp. 20- 24.

[5] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing Shared Resource Contention in Multicore Processors via Scheduling," ASPLOS, 2010, pp. 129-142.

[6] L. Tang, J. Mars, N. Vachharajani, R. Hundt, and M. Soffa, "The Impact of Memory Subsystem Resource Sharing on Datacenter Applications," ISCA, 2011.

[7] L. Zhao, R. Iyer, M. Upton, and D. Newell, "Towards hybrid last level caches for chip-multiprocessors." SIGARCH *Computer Architecture. News*, vol. 36, no. 2, 2008, pp. 56-63.

[8] E. Ebrahimi, C. Joo Lee, O. Mutlu, and Y. N. Patt, "Fairness via source throttling: a configurable and high-performance fairness substrate for multi-core memory systems," SIGARCH *Computer Architecture News*, vol. 38, no. 1, 2010, p.p. 335-346.

[9] M. Ghosh, R. Nathuji, M. Lee, K. Schwan, H.-H. S. Lee, "Symbiotic Scheduling for Shared Caches in Multi-core Systems Using Memory Footprint Signature," In proceedings of International Conference on Parallel Processing, 2011.

[10] A. Jaleel, H. H. Najaf-abadi, S. Subramaniam, S. C. Steely, and J. Emer, "CRUISE: cache replacement and utility-aware scheduling," In Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XVII), 2012, pp. 249-260.

[11] R. B. Myerson, *Game theory: Analysis of conflict*, Harvard University Press, 1991.

[12] Ch. Hauert, and G. Szabó, "Prisoner´s dilemma and public goods games in different geometries: compulsory versus voluntary interactions," *Complexity*, vol. 8, no. 4, 2003, pp. 31-38.

[13] P. Brañas-Garza, and MP. Espinosa, "Unraveling Public Good Games," *Games*, vol. 2, no. 4, 2011, pp. 434-451.

[14] O. Kim, and M. Walker, "The free rider problem: Experimental evidence," *Public Choice*, vol. 43, 1984, pp. 3-24.

[15] 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.

[16] R. Boyd, and P.J. Richerson, "The evolution of reciprocity in sizeable groups," *Theoretical Biology*, vol. 132, no. 3, 1988, pp. 337-356.

[17] G. Szabò, and C. Hauert, "Phase transitions and volunteering in spatial public goods games," *Phys. Rev. Lett.*, vol. 89, no. 11, Sep. 2002, pp. 101–118.

[18] C. Hauert, "Spatial effects in social dilemmas," *Theoretical Biology*, vol. 240, no. 4, Jun. 2006, pp. 627–636.

[19] M. A. Janssen, and R. L. Goldstone, "Dynamic-persistence of cooperation in public goods game when group size is dynamic," *Theoretical Biology*, vol. 243, no. 1, Nov. 2006, pp. 134–142.

[20] 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.

[21] 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.

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

[23] G. Ch. Sirakoulis, I. Karafyllidis, and A. Thanailakis, "A cellular automaton model for the effect of population movement and vaccination on epidemic propagation," *Ecological Modelling*, vol. 133, no. 3, 2000, pp. 209-223.

[24] M.-A. I. Tsompanas, and G.Ch. Sirakoulis, "Modeling and hardware implementation of an amoeba-like cellular automaton," *Bioinspiration&Biomimetics*, vol. 7, 036013 (19pp), 2012.

[25] P. Progias, and G. Ch. Sirakoulis, "An FPGA Processor for modelling wildfire spreading,"*Mathematical and Computer Modelling*, vol. 57, no. 5-6, 2013, pp. 1436–1452.

[26] G.Ch. Sirakoulis, and I. Karafyllidis, "Cellular Automata and Power Consumption," *Journal of Cellular Automata*, vol. 7, no. 1, 2012, pp. 67-80.

[27] K. Ioannidis, G. Ch. Sirakoulis, and I. Andreadis, "Cellular Automata-based Architecture for Cooperative Miniature Robots," accepted for publication in *Journal of Cellular Automata*, 2013