Simulation and Validation of a Heuristic Scheduling Algorithm for Multicore Systems

James Docherty, Alex Bystrov, Alex Yakovlev
School of EEE
Newcastle University
Newcastle-upon-Tyne, UK
james.docherty/a.bystrov/alex.yakovlev@ncl.ac.uk

***Abstract*—The number of embedded systems used worldwide is increasing rapidly. With each generation of equipment, consumers are expecting more computational power and functionality, meaning current designs can be considered unsuitable. As transistor feature size reaches its atomic limit, manufacturers have moved from single to multi-core environments to bridge the performance gap and continue to meet Moores Law. However, this means job scheduling has become exponentially more complex and is reaching a point where standard algorithms are failing to cope. This paper summarizes the initial work performed creating a heuristic based algorithm that is aware of both requests and available resources and therefore is capable of managing the uncertainty brought about by these factors. The work uses Monte Carlo Simulation combined with multi-vary analysis to identify primary contributors and their contribution to scheduling.**

Keywords-Energy Harvesting; Heuristic Algorithms; Monte Carlo Simulations;

I. INTRODUCTION

With the increase in pervasive computing across the world, ever more embedded systems work primarily on stored energy to perform computationally intensive tasks. This means management of both jobs and power becomes more important in every design iteration. A major technique for managing dynamic power dissipation (power consumed by active switching) that is becoming widely used is Dynamic Voltage Frequency Scaling (DVFS). This reduces power consumption at the penalty of increasing execution time. In a system where task execution is short, but with regular activation times, this technique could reduce power consumption by up to 30% [1]. However, many multi-core systems now operate close to their critical voltage, meaning reduction of core voltage to reduce energy consumption is not possible [2]. Therefore more intelligent management of the jobs presented must take place to maximize functionality while conserving reliability.

As performance continues to be a key metric, microprocessor designers aim to meet the goals set by Gordon Moore in 1965 that the number of transistors on an integrated circuit would double every 18 to 24 months [3]. This goal, now known as "Moores Law" is now seen by designers as a target to be met and has been achieved by a number of processors [4]. While this was originally done by decreasing feature size, as transistors head below 10nm, we are approaching the atomic limit; where leakage current is too great to be acceptable for normal use. Therefore, the design of systems rather than devices has changed, with multi core environments allowing Moores predictions to remain true. However, as the number of cores increase, the complexity of managing them grows exponentially. Standard scheduling, working on such methods as First Come First Served or Priority Queuing can cope while the core number is small, but will eventually fail, especially once threads with precedence and relations are present [5]. Eventually these problems will become NP-Hard and feasible schedules will fail to be constructed in a timely manner.

Heuristic Algorithms may offer a solution to this problem, by having simple rules based on factors such as job arrival rate and available energy and using designer intuition,. One such method of heuristic planning is Game Theory which, since its development by Von Neumann in the 1940s [6] and subsequent work by John Forbes Nash [7], has found use in many fields from computer science to evolutionary biology, as well as power management [8] [9].

The models developed tend to take on simple rules, which over time lead to a stabilized outcome. Ideally this outcome will maximize for all players, making the game pareto optimal. This way, if a sudden change occurs, the game will become unstable and eventually settle over a set of repetitions. These games can be modelled as either cooperative or non-cooperative, with much research concentrating on non-cooperative models where every player is competing to maximize their payout. However, some work [10] suggests cooperative games could also lead to greater power savings against the penalty of a power management kernel. Within a system, this game would have many players and input variables, therefore a method must be developed that can reduce these into an optimal model and manage scheduling based on key parameters only.

This paper presents the simulation testing of a Heuristic Algorithm developed in MATLAB to compensate for these issues. The Hypothesis is that a simple algorithm aware of parameters such as overall system load should outperform standard queuing paradigms and increase operational life. Section II details the processor design, queuing methods and simulations conducted, along with techniques used to refine the Heuristic Algorithm. Section III analyses the outcomes from the MATLAB simulation and optimisation runs, discussing the findings. Section IV summarises findings and Section V proposes further investigation for these outcomes.

II. METHODOLOGY

A. Job Queuing

To develop a heuristic based algorithm, a simplified system was modelled in MATLAB. This took the form of Figure 1, with jobs arriving and being sent to cores by a scheduler.

 ISBN: 978-1-61208-308-7

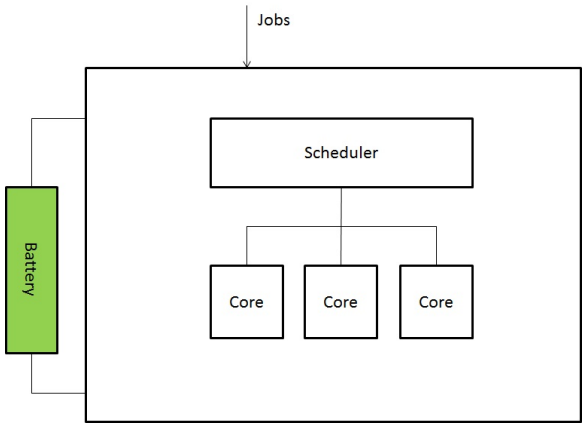

Fig. 1. Simplified Microprocessor Design for Simulations

The cores would manage these jobs and inform the scheduler when they are available to accept new tasks. Three models were chosen for testing. These were:

- First Come First Served (FCFS)
- Priority Queuing
- Heuristic Algorithm under investigation.

First Come First Served is the simplest, but often most effective method of queue management [11]. One queue exists, which all jobs enter and the job at the head of the queue is serviced. This is feasible to implement, while also giving good performance for low workloads. However, as the workload increases, queues can build to extreme lengths and lead to issues.

Priority Queuing adds a second queue for high priority jobs. This is similar to the system seen at airports where check-in desks will serve the main queue until someone joins the fast-track queue. This reduces issues with FCFS when jobs have varying priorities, but can lead to large volumes building up in the main queue if priority jobs continue to block the processor either through high arrival rates or processor utilization.

Both these and other scheduling algorithms fail to dynamically manage cores to maximize system reliability. Many also lack the ability to manage precedence and are unaware of energy as an input. The proposed solution in this paper provides these important items, while also managing job direction through simple heuristics, rather than a complex management program.

B. Literature Review

Heuristics have proved a popular method of managing jobs in a variety of areas, especially where a level of non-determinism is present, as this can lead to exhaustive search methods becoming excessively long in their computation time. Methods can use explicit rules to find optimal strategies [12] or more abstract techniques such as Nash Equilibrium discussed previously [13]. Heuristics allow for schedules that are clearly infeasible to be ignored through initial grouping, which reduces computation time and can improve performance [14] [15]. This use of heuristics means some scheduling that would originally have been performed offline can now take place in real-time, improving the reliability of operation [16]. As complexity of systems increases, the use of rule-based schedulers will become more commonplace. While these cannot always give the best response, the outcome will often be suitable in the time taken to calculate it that no detriment to the quality of service will perceptibly take place [17].

C. System Level Simulation

An algorithm, which can be seen in Algorithm 1, was created to allow System Level Simulations to take place. This let a series of jobs be run through a Monte Carlo Simulation. Jobs are created based on a rate (λ) and processed based on another rate (μ). One to three cores can also be implemented, allowing the difference in queue times, queue length and processor utilization to be observed. For this jobs are assumed to be arriving from activation (t_0) at a rate of $\lambda e^{-\lambda t}$ and processed at a rate of $\mu e^{-\mu t}$. The simulation is run over 72000 cycles with 10 repeats for each case and λ/μ values of 10, 30 and 50. One, two and three cores are active in versions of the simulation and all simulations are repeated ten times to give consistency of results.

Once job arrival and service rates are determined in lines 4-5, the system enters the 72000 operational cycles. The algorithm considers available energy prior to scheduling and adjusts the level to activate the Heuristic section through variable $MaxQueueLength$. Cores are only activated to service jobs if the current queue length is greater than this variable, preventing the excessive consumption of energy but increasing total operation time for jobs. If all cores are active and a job is low priority, it will be placed into the queue at the tail, replicating FCFS. However, priority jobs will be placed into the priority queue, which will be serviced by the next available core. All data was outputted to a CSV file on completion of 72000 cycles or if system energy reached zero during simulation when this was considered.

Data from the simulations was collected and analysed using Minitab, a statistical program used for data analysis, to look for statistical differences between scheduler types and key parameters to be used in any improvement exercises.

D. Addition of Energy

Once these experiments were complete, giving a baseline of algorithm effectiveness, the extra uncertainty of energy was added to the simulations. This was done by placing a battery into the simulation with a percentage of charge. The FIFO and Priority Queues were unaware of this and therefore continued executing jobs at the same rate until failure. However, as the state of charge decreased, the aggression of the heuristic algorithm in activating cores decreased through $MaxQueueLength$, meaning the execution time for jobs increased. While this would seem to decrease the quality of service, the goal is to maximise lifetime of the system. Since fewer cores are activated, both the static and dynamic energy consumptions are decreased — thereby increasing the operational life of the system. Priority queuing is still active

 ISBN: 978-1-61208-308-7

Algorithm 1 Simulation for Heuristic Scheduler

Set λ and μ
Set Flags and Clock to zero, Energy to 100
for $ArrivalTime$ and $ServiceTime = 1$ to 72000 **do**
  $ArrivalTime(n) = \lambda e^{-\lambda t}$
  $ServiceTime(n) = \mu e^{-\mu t}$
end for
while $Clock < 72000$ **do**
  if $Energy < 50$ **then**
    $MaxQueueLength = 2$
  else
    $MaxQueueLength = 4$
  end if
  for $AllActiveCores$ **do**
    $ServiceTime - -$
  end for
  if $QueueLength > 0$ **then**
    if $CoreisEmpty$ & $CoreIsActive$ **then**
      $PlaceJobonCore$
    else if $CoreisEmpty$ & $CoreisInactive$ & $QueueLength > MaxQueueLength$ **then**
      $ActivateCore, PlaceJob$
    end if
    if $Clock = ArrivalofNewJob$ **then**
      if $CoreisEmpty$ & $CoreisActive$ **then**
        $PlaceJobonCore$
      end if
    else if $CoreisEmpty$ & $CoreisInactive$ & $QueueLength > MaxQueueLength$ **then**
      $ActivateCoreandPlaceJob$
    else if $AllCoresAreBusy$ & $JobIsNonpriority$ **then**
      $PlaceJobinQueue$
      $QueueLength + +$
    else if $AllCoresAreBusy$ & $JobIsPriority$ **then**
      $PlaceJobinPriorityQueue$
    end if
  end if
  for $EachActiveCore(1 - x)$ **do**
    $Core(x)EnergyConsumed = \mathcal{U}(0.002, 0.005)$
  end for
  $Energy = Energy - AllCore(x) EnergyConsumed - \mathcal{N}(0.01, 0.005)$
  if $Energy <= 0$ **then**
    Break
  end if
  $UpdateWaitTime, IdleTime, QueueLength$
  $Clock + +$
end while
print Value of Energy, Cores, λ, μ, Average Wait Time, Max Wait Time, Average Queue Length, Max Queue Length, Clock, Cores 1-3 Idle Percentage

and priority jobs will be fast tracked to the core, with jobs currently occupying it halted. Once all priority tasks are cleared from the system, the non-priority jobs may resume executing. In extreme cases, the scheduler may activate a core to push through more priority jobs, thus maintaining quality of service for a small energy penalty. Though this will shorten the operational life of the entire system, missing priority jobs could be hazardous, especially in real-time or safety critical systems. Therefore this reduction in system lifetime is justified by keeping key systems active.

E. Design of Experiments

Following this preliminary work, a Design of Experiments (DoE) was conducted to determine each variables overall contribution and whether any interaction between variables took place. Within a DoE, key parameters and their values are run to identify which have a major effect and which can be deemed insignificant[18]. All possible levels of interaction are initially investigated, with insignificant higher-orders removed until a minimized model exists. This model can then be mathematically analysed by a generalization such as the General Linear Model (GLM) to give the percentage contribution (ϵ) that each parameter, including error, delivers to the overall system. The DoE was set up in Minitab as a two-level, four input, full factorial DoE with five repeats, giving 80 data points ($5\text{x}2^4$). Parameters were as follows:

- Level of Battery to increase Queue Length for Core Activation = 30/60
- Kick Out of non-priority jobs for priority jobs = Off/On
- λ=10/50
- μ=10/50

These results were analysed to determine key parameters for the GLM to determine the contributions each factor gives to the overall effect. These factors can allow further testing to give ideal results and an optimal scheduler design for the Heuristic Environment.

III. RESULTS

For the simulations undertaken, Table I shows the outcome for tests conducted where $\lambda = \mu$ and energy was considered. For this cycle, the Heuristic Algorithm outperforms both FCFS and Priority Queuing by a factor of 3, giving a statistically significant result. In cases where $\lambda \geq \mu$, the Heuristic Algorithm consistently outperforms its rivals and also gives significantly longer operating life in situations where $\lambda < \mu$ due to the dynamic deactivation of superfluous cores. This result is shown for other values of $\frac{\lambda}{\mu}$ in Fig 2, demonstrating that the Heuristic Scheduler outperforms both FCFS and Priority Scheduling for a range of values. No detriment to lifetime occurs with an increase in load rate, as proved by a regression test on the Heuristic Algorithm results (P=0.507, therefore no correlation between job rate and clock cycles completed).

When tests were conducted with single and dual core architectures the results, seen in Fig 3, show the Heuristic Algorithm continued to outperform both FCFS and Priority Queuing. With only one core active, Heuristic methods give

 ISBN: 978-1-61208-308-7

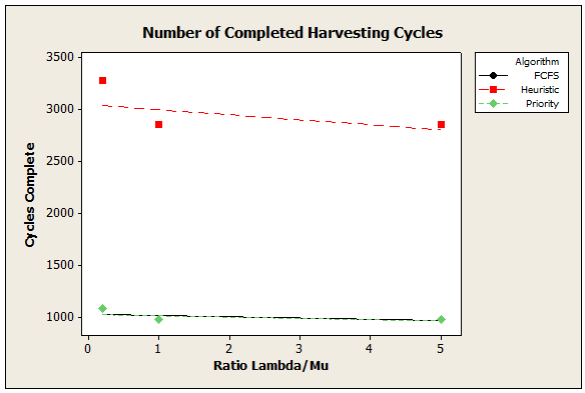

Fig. 2. Scatterplot showing Clock Cycles Complete for increasing values of lambda over mu

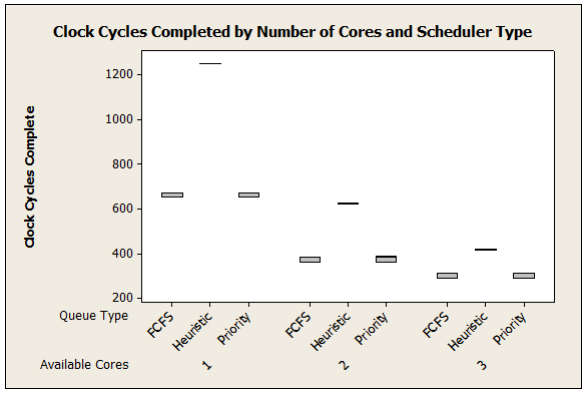

Fig. 3. Boxplot of Microprocessor Life for varying number of cores

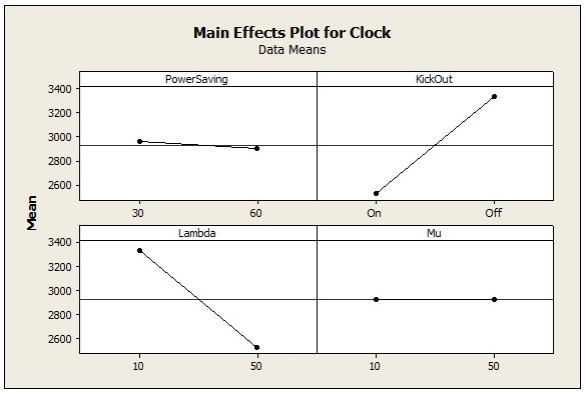

Fig. 4. Main Effects Plot of DoE

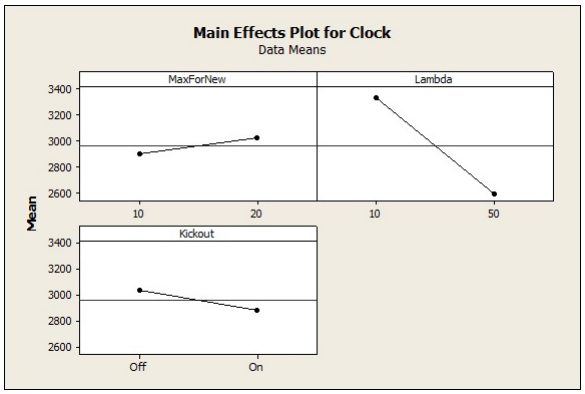

Fig. 5. Main Effects Plot showing additional parameters to Fig 4

a factor of two increase in operational lifetime due to the dynamic management of cores and their deactivation during light loading — similar to the sleep mode present in many modern microprocessors. As the core count increases, the effectiveness of the Heuristic Management can still be seen, giving improvements of 61 % for two cores and 35 % for three. This reduction in effectiveness is due to the increased dynamic and leakage power consumption present for a greater number of cores.

TABLE I
ONE-WAY ANOVA RESULTS FOR NUMBER OF CLOCK CYCLES COMPLETED COMPARED TO QUEUING METHOD WHEN $\lambda = \mu$

Queueing Method	Mean	Standard Deviation	P-Value	Conclusion
FCFS	971.00	0.25	0.00	Heuristic Outperforms
Priority Queue	970.96	0.32		
Heuristic	2848.79	0.54		

Following analysis of the DoE, Fig 4 shows an increase in Lambda, as well as deactivation of the Kick Out function have a significant effect on the number of successful clock cycles completed. Further analysis, shown in Fig 5, reveals that increasing the value of $MaxQueueLength$ in Algorithm 1 gives some increase to operational life and concurs with Fig 4 that being able to remove tasks from processors to allow execution of priority jobs (the "kickout" function) reduces operational life of the system – as this will increase the utilisation of processors and thus affect energy consumption. Due to this, a second analysis only looking at priority jobs was undertaken and can be seen in Fig 6. For this, deactivating Kick Out can be seen to have a large effect on missed priority jobs as these are now made to wait until a core has completed execution, rather than allowing the job immediate access.

The results from these experiments were placed into a GLM within Minitab to see determine main contributions to the

TABLE II
EPSILON SCORES FOR GENERAL LINEAR MODEL FOLLOWING DOE

Source	P-Value	ϵ
Power Saving	0.000	0.152
Kick Out	0.000	33.162
Lambda	0.000	33.097
Mu	0.756	0.000
Power Saving*Kick Out	0.000	0.153
Power Saving*Lambda	0.000	0.156
Power Saving*Mu	1.00	0.000
Kick Out*Lambda	0.000	33.125
Kick Out*Mu	0.022	0.000
Lambda*Mu	0.165	0.000
Power Saving*Kick Out*Lambda	0.000	0.155
Error		0.000
Total		100.0

 ISBN: 978-1-61208-308-7

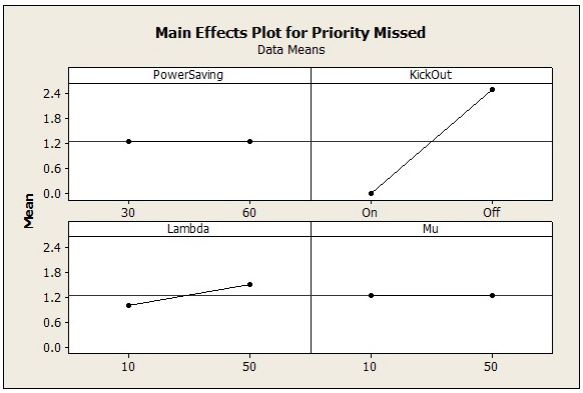

Fig. 6. Main Effects Plot analysing Priority Jobs only

microprocessor lifetime. The results for this, shown in Table II clearly show the contribution the kickout function and lambda have on the system, accounting for 99 percent of the effect seen. For these runs, error can be seen to be zero, due to the level of non-determinism afforded by the simulation being low. For repeat runs on a real microprocessor, this value would be expected to rise significantly.

IV. Conclusions

This work shows that under low loads ($\lambda < \mu$), a first come first served scheduler is capable of managing all jobs easily. With only one core active, FCFS gives a high value of maximum wait time compared to average wait time. Therefore this would not be suitable for a system with priority jobs. With two cores active however, this wait time reduces dramatically and priority queuing may not be required. However, when $\lambda > \mu$, priority queuing and multiple active cores becomes a feasible way to manage jobs.

The heuristic algorithm presented provides the flexibility of both strategies, combined with dynamic core management and therefore increased energy efficiency. By altering the queue length required to activate a new core with respect to energy available the algorithm, this extra non-deterministic aspect can be managed and the quality of service for an end user maintained.

When $\lambda \gg \mu$, a drop in performance for all scheduling methods takes place. As the arrival rate is so much larger than the service rate, the queue grows exponentially; meaning processor utilisation is always at 100 %. Due to this, no dynamic core management can take place and all schedulers perform at a comparable rate. Within these simulations, this leads to FCFS and Priority Queuing outperforming the Heuristic Algorithm, as the model designed considered the extra complexity required and increased leakage power accordingly — causing lifetime for the Heuristic Algorithm to be reduced.

While this system has only been tested up to three cores, it is thought that the algorithm would feasibly cope with a larger number of available devices in its current design. Since the scheduler operates as an overseer for all devices, it simply places jobs onto the first core it finds available; or activates a core if required. A limitation for this is the simulation work required to determine suitable values for $MaxQueueLength$ and the Energy at which to alter this. This work also currently presumes a homogeneous layout, but it is expected that heterogeneous microprocessors (where cores of different design are placed on the same silicon) will become more commonplace in the near future.

While this work only addresses a generic system with general job types, some further tests with real-time jobs have been conducted in a single-core variant of this work [19] and found to give an improvement in reliability over standard schedulers such as FCFS. This work developed an automotive Electronic Control Unit (ECU) within the MATLAB environment and had multiple priority levels for jobs. This further validates the work presented in this paper and shows the use of Heuristics in practical environments can give operational benefits.

V. Future Work

This paper proves the concept of a heuristic algorithm is viable and can dynamically manage a multi-core stored energy environment with minimal complexity. Key factors in its management have been identified and initially tested through simulation, which has allowed rapid testing of the many permutations and validation of the concept. The progression of this work is to conduct a Design of Experiments (DoE) to optimize the algorithm through determining each input variables overall contribution to both wait time and system reliability. DoEs have been used in previous work to great effect in identifying key items within an energy harvesting environment [20].

Owing to computational restrictions, only architectures up to three cores were investigated in this work. With the number of cores predicted to reach more than 300 by 2020 [21], tests of massive many-core layouts would be necessary to prove the versatility and usefulness of the Heuristic Algorithm against other more-advanced scheduling paradigms. While MATLAB is a useful tool for performing this, the runtimes for a 300 core simulation on a desktop computer would be excessive. One solution to this would be the use of a grid computer such as HTCondor to run the Monte Carlo simulation across a distributed system; thus reducing the test time and allowing validation of this concept on a massive many-core architecture.

As scalability may be an issue for heuristic algorithms, investigations into the use of pruning and filtering techniques are currently on going. These would give a hybrid approach, where certain situations would be managed by a standard schedule design, saving the use of heuristics for intensive or high-difficulty cases. Through this method, it is thought that the heuristic design could be preserved and used on more complicated systems without the need for redesign at each iteration.

While conceptually the design can be construed as sound, tests in a practical multi-core architecture will determine whether the algorithm works practically. Investigation for a suitable real-world processor and development kit are underway, with plans to port the algorithm into the system kernel

 ISBN: 978-1-61208-308-7

and perform tests against established schedulers including FCFS.

REFERENCES

[1] W. H. Wolf, *Modern VLSI design : systems on silicon*, 2nd ed. Upper Saddle River, NJ: Prentice Hall PTR, 1998.

[2] A. Bartolini, M. Cacciari, A. Cellai, M. Morelli, and A. Tilli, "Fault Tollerant Thermal Management for High-Performance Multicores," in *Workshop on Micro Power Management for Macro Systems on Chip as part of DATE 11*, D. Marculescu and S. Lesecq, Eds., Grenoble, France, 2011, accessed: Aug 5 2013. [Online]. Available: http://users.ece.cmu.edu/ dianam/uPM2SoC10/

[3] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, pp. 114–117, 1965.

[4] L. Torres, P. Benoit, G. Sassatelli, M. Robert, D. Puschini, and F. Clermidy, *An Introduction to Multi-Core System on Chip Trends and challenges*, 1st ed., M. Hubner and J. Becker, Eds. New York: Springer, 2011.

[5] A. S. Tanenbaum, *Modern Operating Systems*, 3rd ed. Upper Saddle River, N.J.: Pearson/Addison Wesley, 2009.

[6] J. Von Neumann and O. Morgenstern, *Theory of games and economic behavior*, 60th ed. Princeton, N.J. ; Woodstock: Princeton University Press, 2007.

[7] J. F. Nash, "Non-Cooperative Games," *The Annals of Mathematics*, vol. 54, no. 2, pp. 286–295, 1951.

[8] B. Foo and M. van der Schaar, "A Queuing Theoretic Approach to Processor Power Adaptation for Video Decoding Systems," *Signal Processing, IEEE Transactions on*, vol. 56, no. 1, pp. 378–392, 2008.

[9] A. Wierman, L. L. H. Andrew, and T. Ao, "Stochastic analysis of power-aware scheduling," in *Communication, Control, and Computing, 2008 46th Annual Allerton Conference on*, 2008, pp. 1278–1283.

[10] S. Hsien-Po and M. van der Schaar, "Conjecture-based channel selection game for delay-sensitive users in multi-channel wireless networks," in *Game Theory for Networks, 2009. GameNets '09. International Conference on*, 2009, pp. 241–250.

[11] D. Gross, *Fundamentals of queueing theory*, 4th ed. Hoboken, N.J.: Wiley, 2008.

[12] Y. Nomura, M. Iwamoto, T. Yamanouchi, and M. Watanabe, "A heuristics guided scheduling framework for domains with complex conditions," *Proceedings Sixth International Conference on Tools with Artificial Intelligence. TAI 94*, pp. 752–755, 1994, accessed: 5 August 2013. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=346409

[13] I. Ahmad, "Using game theory for scheduling tasks on multi-core processors for simultaneous optimization of performance and energy," *2008 IEEE International Symposium on Parallel and Distributed Processing*, pp. 1–6, Apr. 2008, accessed: 5 August 2013. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4536420

[14] H. Cheng, "A High Efficient Task Scheduling Algorithm Based on Heterogeneous Multi-Core Processor," *2010 2nd International Workshop on Database Technology and Applications*, no. 3, pp. 1–4, Nov. 2010, accessed: 5 August 2013. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5659041

[15] D. Dal and N. Mansouri, "Power Optimization With Power Islands Synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 7, pp. 1025–1037, Jul. 2009, accessed: 5 August 2013. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5075810

[16] B. Zhao, H. Aydin, and D. Zhu, "Reliability-Aware Dynamic Voltage Scaling for Energy-Constrained Real-Time Embedded Systems," in *Computer Design, 2008. ICCD 2008. IEEE International Conference on*, vol. 546244, 2008, pp. 633–639.

[17] G. C. Buttazzo, *Hard real-time computing systems : predictable scheduling algorithms and applications*, 2nd ed. New York: Springer, 2005, accessed: 5 August 2013. [Online]. Available: http://www.loc.gov/catdir/toc/fy0613/2004058935.html

[18] F. W. Breyfogle Iii, *Implementing Six Sigma : Smarter Solutions Using Statistical Methods*, 2nd ed. Hoboken: John Wiley & Sons Inc., 2003.

[19] J. Docherty, A. Bystrov, and A. Yakovlev, "Simulation Testing of a Real-Time Heuristic Scheduler with Automotive Benchmarks," in *UK SIM 2013*, D. Al-dabass, Ed., Cambridge, UK, 2013.

[20] ——, "Identification of Key Energy Harvesting Parameters Through Monte Carlo Simulations," in *UK SIM 2012*, D. Al-dabass, Ed., vol. 0, Cambridge, UK, 2012, pp. 486–490, accessed: 5 August 2013. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/UKSim.2012.73

[21] S. Borkar, "Thousand Core Chips A Technology Perspective," in *Design Automation Conference*, 2007, pp. 746–749.

 ISBN: 978-1-61208-308-7