Distributed Scheduling for Many-Cores Using Cooperative Game Theory

Anuj Pathania*, Vanchinathan Venkataramani†, Muhammad Shafique*, Tulika Mitra†, Jörg Henkel*
*Chair of Embedded System (CES), Karlsruhe Institute of Technology, Germany
†School of Computing, National University of Singapore, Singapore
Corresponding Author: anuj.pathania@kit.edu

ABSTRACT

Many-cores are envisaged to include hundreds of processing cores etched on to a single die and will execute tens of multi-threaded tasks in parallel to exploit their massive parallel processing potential. A task can be sped up by assigning it to more than one core. Moreover, processing requirements of tasks are in a constant state of flux and some of the cores assigned to a task entering a low processing requirement phase can be transferred to a task entering high requirement phase, maximizing overall performance of the system.

This scheduling problem of partial core reallocations can be solved optimally in polynomial time using a dynamic programming based scheduler. Dynamic programming is an inherently centralized algorithm that uses only one of the available cores for scheduling-related computations and hence is not scalable. In this work, we introduce a distributed scheduler that disburses all scheduling-related computations throughout the many-core allowing it to scale up. We prove that our proposed scheduler is optimal and hence converges to the same solution as the centralized optimal scheduler. Our simulations show that the proposed distributed scheduler can result in 1000x reduction in per-core processing overhead in comparison to the centralized scheduler and hence is more suited for scheduling on many-cores.

CCS Concepts

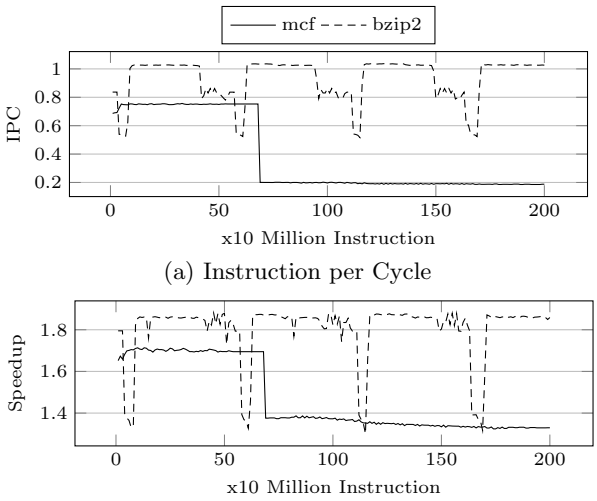
•Computer systems organization → Self-organizing autonomic computing;

Keywords

Many-Core, Distributed Scheduling, Multi-Agent Systems

1. INTRODUCTION

Many-cores are upcoming hundred core processors that execute tens of multi-threaded tasks in parallel. A task can be allocated to more than one core to speed up its processing. *Speedup of a task with N cores is defined as Instruction per Cycle (IPC) of the task when assigned N cores in comparison to the task's IPC when assigned one core in an isolated execution.* In this work, we use a special kind of many-core called **adaptive many-core** [10, 14] that can execute single-threaded tasks on multiple cores. *This also allows the acceleration of both single-threaded and multi-threaded tasks* by exploitation of Instruction Level Parallelism (ILP) and Thread Level Parallelism (TLP), respectively.



(a) Instruction per Cycle

(b) 2-core speedup

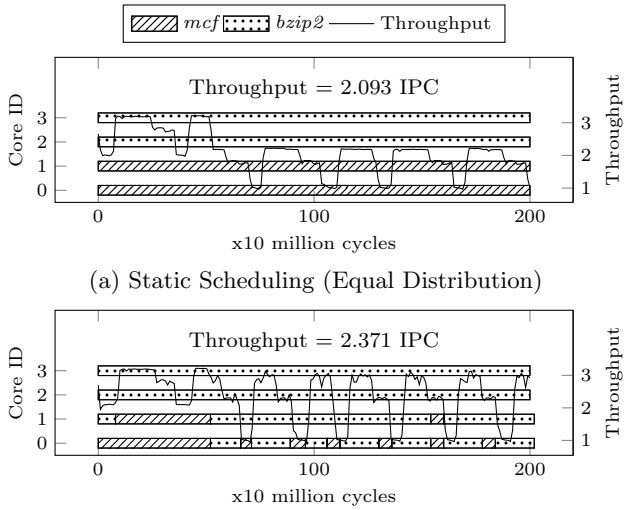
Figure 1: Execution profiles of *mcf* and *bzip2* benchmarks.

Dynamic Scheduling Motivation: Figure 1a shows processing requirements of executing tasks (*mcf* and *bizp2*) vary over time. Further, ILP/TLP exploitation potential in benchmarks also changes during execution resulting in a varying speedup on multiple cores as shown in Figure 1b. To keep the many-core operating at peak performance, cores from task entering a low requirement phase need to be transferred to a task entering a high requirement phase by a scheduler. Figure 2 shows 13.28% gain in throughput (total IPC) can be obtained from performing dynamic partial core reallocations in a 4-core processor running two tasks (*mcf* and *bizp2*) compared to a static approach, where two cores are allocated to each task.

Authors in [7] proposed a Dynamic Programming (DP) based scheduler that can optimally solve the partial core reallocation problem on many-cores in polynomial time. Dynamic programming is an inherently centralized algorithm that uses only one core of the many-core for performing scheduling-related computations and hence cannot scale up.

DAC '16, June 05-09, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2897937.2898009



(a) Static Scheduling (Equal Distribution)

(b) Dynamic Scheduling (Optimal Reallocation)

Figure 2: Dynamic scheduling can provide 13.28% additional throughput over static scheduling.

As an alternative, in this work we propose a distributed scheduler. To perform distributed scheduling, we propose a Multi-Agent System (MAS) scheduler based on Cooperative Game Theory (CGT) called ***CGT-MAS***. CGT is a sub-branch of economic game theory wherein agents can cooperate with each other to form coalitions, and these coalitions then compete with other coalitions. It is intuitively applicable to scheduling on many-cores, wherein cores come together to form core-coalitions on which tasks are executed. *A core-coalition is a virtual construct formed at run-time when multiple cores are assigned to the same task.*

MAS schedulers [6] based on game theory, with foundations in Nash equilibrium have been proposed for many-cores before [1, 19]. But we use a new approach for proving Nash equilibrium in this work that allows *CGT-MAS* to provide stronger formal theoretical results on convergence, quality of converged solution and time to converge than previous attempts. We theoretically prove that CGT-MAS will converge to the same optimal solution as the dynamic programming based scheduler. Hence, CGT-MAS provides scalability without any compromise on the quality of the solution. In a complementary work, we present a distributed fair scheduling solution for many-cores in [13].

Our Novel Contributions: We present a distributed scheduler called *CGT-MAS*, which performs dynamic scheduling on many-cores. *CGT-MAS* disburses its processing overhead across all cores in the many-core enabling its scalability. *CGT-MAS guarantees convergence to the optimal solution similar to dynamic programming based scheduler from any initial state in a finite number of steps but with* 1000x *less per-core processing overhead. Therefore, CGT-MAS is suited for performing task scheduling on many-cores.*

2. SCHEDULING WITH CGT-MAS

System Overview: We now present our proposed MAS scheduler *CGT-MAS*. We assign a unique agent to each core of the many-core. A unique core-coalition is assigned to each task to be executed on the many-core that the agents can join or leave anytime changing the core-coalitions sizes in the process. *We assume our tasks are malleable, which means the number of cores allocated to a task can change during its execution.* We also assume that there is one to many mapping between tasks and cores. This makes our optimization search-space discrete.

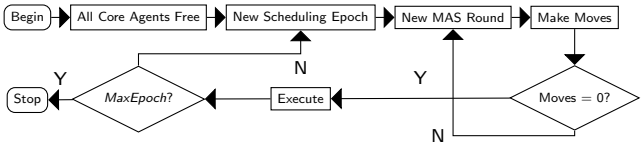


Figure 3: Execution Flow for *CGT-MAS*

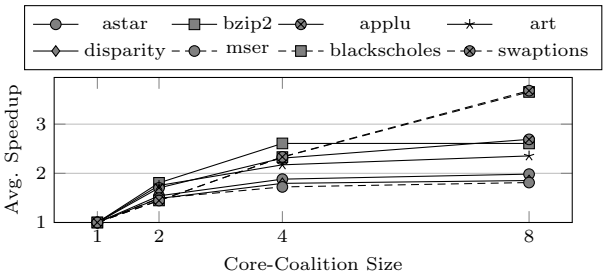


Figure 4: Average speedup of varisized core-coalitions over singular core-coalitions for different tasks.

System Models: We model the problem of task scheduling on many-core in game theoretic notation. Let A represent the set of $|A|$ agents, indexed by a_i. Let C represent the set of $|C|$ core-coalitions, indexed by C_j. $|C_j|$ represent the current size of (or number of agents in) C_j.

We define the quality of service of a task by its IPC. $Q(C_j)$ represents the current IPC of the task associated with C_j. $Q(C_j)$ can vary as the associated task passes through its different execution phases on C_j as shown in Figure 1. We allot a unit value every time an instruction is successfully committed. Thus, $Q(C_j)$ represents the value associated with C_j. $V(C)$ represents the total value generated by all core-coalitions C and is an abstraction for the total number of instructions committed by the many-core. We define current utility u_{a_i} for an agent a_i as the increase in IPC it brings to the core-coalition it has joined. Our utility definition is inspired by the CGT model presented in [2].

$$V(C) = \sum_{\forall C_j \in C} Q(C_j) \quad (1)$$

$$u_{a_i}(C_j) = Q(C_j) - Q(\{C_j - a_i\}) \quad (2)$$

Execution Flow: Figure 3 shows the execution flow of *CGT-MAS*. Initially, all agents are free. At the beginning of each scheduling epoch, a series of MAS rounds take place to determine mapping. *Scheduling epoch is the granularity at which scheduling is performed in the operating system.* In each round, agents evaluate the benefits of joining every other core-coalition they are not part of against the benefits of staying in their current core-coalitions based on their utilities. Agents then myopically take decisions to move amongst core-coalitions to increase their utilities. It suffices for one agent in a given core-coalition to perform utility calculations and take decisions on behalf of all other agents in that core-coalition to reduce overheads. An agent a_i will change core-coalition from C_j to $C_{j'}$ if $u_{a_i}(C_{j'} \cup \{a_i\}) > u_{a_i}(C_j)$. For estimating utilities, agents use IPC prediction techniques for adaptive many-cores developed in [17]. Rounds stop when no more moves are possible. Tasks are then executed with cores allocated to them in the final round. System halts when a user-defined *MaxEpoch* is reached.

Execution Characteristics: Figure 4 shows the average speedup for benchmarks of different types on varisized core-coalitions. It can be observed from the figure that the average speedup is both monotonically increasing and concave. *This is because of the saturation of exploitable ILP or TLP in the benchmarks with increasing core-coalition size.* Still, addition of every core to the core-coalitions brings a non-negative increase in the IPC of the associated benchmark. This by definition makes the task's IPC **non-decreasing** and agent's utility **sub-modular**.

$$Q(C_j) \geq Q(C'_j) \text{ if } |C_j| \geq |C'_j| \quad (3)$$

$$Q(C_j) - Q(\{C_j - a_i\}) \leq Q(C'_j) - Q(\{C'_j - a_i\}) \text{ if } |C_j| \geq |C'_j| \quad (4)$$

Equilibrium (Stability): Stability with respect to *CGT-MAS* means that if the system load does not change, then the system will achieve oscillation-free tasks-to-cores mapping. When stable, no agent has an incentive to move - a state of *Nash* equilibrium.

$$\forall_{(a_i \in (C_j)) \in C} \quad \nexists_{C_{j'} \in C} \quad u_{a_i}(C_{j'} \cup \{a_i\}) > u_{a_i}(C_j) \quad (5)$$

We choose to prove equilibrium using *Sharkovsky* theorem [5], by denying the existence of a two-period cycle in our system.

Let $C_x \rightleftharpoons C_y$ represent core-coalitions C_x and C_y in equilibrium. Let a_x and a_y represent agents that are part of core-coalitions C_x and C_y, respectively. Based on Equations (2) and (5) following equations hold.

$$Q(C_x) - Q(\{C_x - a_x\}) \geq Q(C_y \cup \{a_x\}) - Q(C_y) \quad (6)$$

$$Q(C_y) - Q(\{C_y - a_y\}) \geq Q(C_x \cup \{a_y\}) - Q(C_x) \quad (7)$$

Lemma 1. *In equilibrium $C_x \rightleftharpoons C_y$, if a_x does not want to move, then subset $\{a_x, a_{x'}\} \subseteq C_x$ will also not move.*

Proof. From Equation (4) we know,

$$\begin{aligned} Q(C_x - \{a_x\}) - Q(C_x - \{a_x, a_{x'}\}) &\geq Q(C_y \cup \{a_x, a_{x'}\}) - Q(C_y \cup \{a_x\}) \\ Q(C_x) - Q(C_x - \{a_x, a_{x'}\}) &\geq Q(C_y \cup \{a_x, a_{x'}\}) - Q(C_y) [\because \text{Eq. (6)}] \end{aligned}$$

This result can be extended to any subset size and implies that results shown for a single agent will also extend to subset of agents in a core-coalition; hence proved. □

Lemma 2. *If a new agent a_z is added to equilibrium $C_x \rightleftharpoons C_y$, then equilibrium continues to hold.*

Proof. Without loss of generality, let us say utility for agent a_z is greater on joining C_x than C_y and hence agent a_z joins C_x. By this assumption following equation is true,

$$Q(C_x \cup \{a_z\}) - Q(C_x) \geq Q(C_y \cup \{a_z\}) - Q(C_y)$$

By adding and subtracting a_z from C_x we get,

$$Q(C_x \cup \{a_z\}) - Q(C_x \cup \{a_z\} - \{a_z\}) \geq Q(C_y \cup \{a_z\}) - Q(C_y)$$

After joining C_x, a_z and $a_x \in C_x$ have equal utility,

$$Q(C_x \cup \{a_z\}) - Q(C_x \cup \{a_z\} - \{a_x\}) \geq Q(C_y \cup \{a_x\}) - Q(C_y)$$

Therefore, a_x would not move from C_x even after a_z joins. From Equation (4) we know,

$$\begin{aligned} Q(C_x \cup \{a_y\}) - Q(C_x) &\geq Q(C_x \cup \{a_z, a_y\}) - Q(C_x \cup \{a_z\}) \\ Q(C_y) - Q(C_y - \{a_y\}) &\geq Q(C_x \cup \{a_z, a_y\}) - Q(C_x \cup \{a_z\}) [\because \text{Eq. (7)}] \end{aligned}$$

Therefore, $a_y \in C_y$ would also not move; hence proved. □

Lemma 3. *If an agent $a_{x'} \in C_x$ is removed from equilibrium $C_x \rightleftharpoons C_y$, it is reattained in no more than one move.*

Proof. Without loss of generality, this result would hold even if an agent was removed from C_y. From Equation (4),

$$\begin{aligned} Q(C_x - \{a'_x\}) - Q(C_x - \{a'_x, a_x\}) &\geq Q(C_x) - Q(C_x - \{a_x\}) \\ Q(C_x - \{a_{x'}\}) - Q(C_x - \{a_{x'}, a_x\}) &\geq Q(C_y \cup \{a_x\}) - Q(C_y) [\because \text{Eq. (6)}] \end{aligned}$$

Therefore, a_x would not move from C_x even after $a_{x'}$ leaves.

Now a move is possible from C_y to C_x since after leaving of $a_{x'}$ the utility of joining C_x has increased. We assume an agent $a_{y'} \in C_y$ makes that move. From Equation (7),

$$\begin{aligned} Q(C_y) - Q(C_y - \{a_y\}) &\geq Q(C_x \cup \{a_y\}) - Q(C_x) \\ Q(C_y - \{a_{y'}\}) - Q(C_y - \{a_{y'}, a_y\}) &\geq Q(C_y) - Q(C_y - \{a_y\}) [\because \text{Eq. (4)}] \end{aligned}$$

Since, C_x is equivalent to $(C_x - \{a_{x'}\}) \cup \{a_{y'}\}$ we obtain

$$\begin{aligned} Q(C_y - \{a_{y'}\}) - Q(C_y - \{a_{y'}, a_y\}) \geq\ & Q(C_x - \{a_{x'}\} \cup \{a_{y'}, a_y\}) \\ & - Q(C_x - \{a_{x'}\} \cup \{a_{y'}\}) \end{aligned}$$

Therefore, a_y would not move from C_y; hence proved. □

Lemma 4. *If a core-coalition C_z is added to equilibrium $C_x \rightleftharpoons C_y$, an equilibrium $C_x \rightleftharpoons C_y \rightleftharpoons C_z$ will be attained.*

Proof. Without loss of generality, let us assume only two core-coalitions interact at a time, beginning with C_x and C_z. By Lemma- 2 and 3 when C_x and C_z unidirectionally exchange agents until equilibrium $C_x \rightleftharpoons C_z$ is reached, equilibrium $C_x \rightleftharpoons C_y$ will continue to hold. After $C_x \rightleftharpoons C_z$ is reached, C_y and C_z unidirectionally exchange agents until equilibrium $C_y \rightleftharpoons C_z$ is reached, while $C_x \rightleftharpoons C_y$ holds. Thereby, $C_x \rightleftharpoons C_y \rightleftharpoons C_z$ is attained; hence proved. □

Theorem 1. *Our MAS will achieve oscillation-free equilibrium from any given initial state in $O(C)$ rounds.*

Proof. From any given initial state, any core-coalition is trivially in equilibrium when considered in isolation. Equilibrium can be iteratively extended using Lemma 4 to any number of core-coalitions. Further, Lemmas 2 and 3 show that cycles of period two cannot exist in our MAS because exchange of agents between core-coalition is always unidirectional. Since period two cycle is easiest to create, corollary of Sharkovsky theorem [5] says that in a dynamic system if cycle of period two does not exist then cycle of any higher period also does not exists. Additionally, since no cycle exists, an agent cannot return to core-coalition it had previously left. Thus, any agent can make a maximum of $|C|$ jumps (excluding incorrect myopic movement corrections) before system reaches the equilibrium; hence proved. □

Theorem 2. *In equilibrium state on a many-core, agent allocation is optimal under predicted information.*

Proof. In equilibrium $C_x \rightleftharpoons C_y$, let us assume $V(C)$ is not optimal under the purview of predicted IPC, and there exists another allocation of agents to core-coalitions C'_x and C'_y that is optimal. From Equation (1) we know,

$$Q(C'_x) + Q(C'_y) > Q(C_x) + Q(C_y) \quad (8)$$

We assume all agents are part of either of the core-coalitions and without loss of generality (Lemma 1) we say,

$$C'_x = C_x - \{a_x\} \text{ and } C'_y = C_y \cup \{a_x\}$$

Now since $C_x \rightleftharpoons C_y$, from Equation 6 we have

$$\begin{aligned} Q(C_x) - Q(\{C_x - a_x\}) &\geq Q(C_y \cup \{a_x\}) - Q(C_y) \\ \implies Q(C_x) - Q(\{C'_x\}) &\geq Q(C'_y) - Q(C_y) \\ \implies Q(C_x) + Q(C_y) &\geq Q(\{C'_x\}) + Q(C'_y) \end{aligned}$$

a contradiction to Equation (8); hence proved. □

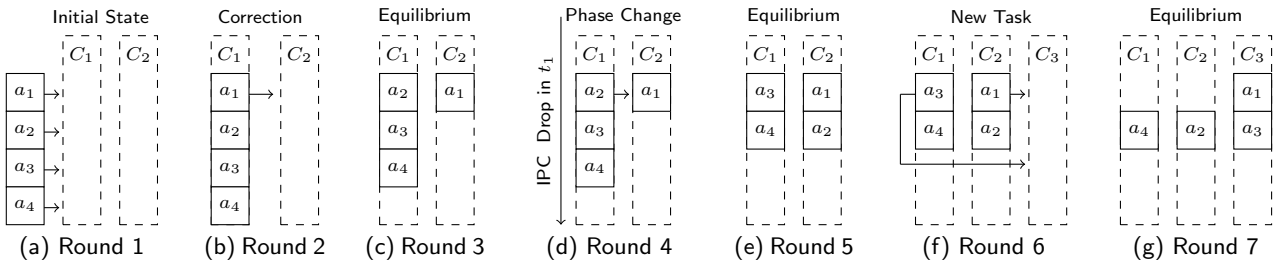


Figure 5: Illustrative example demonstrating the Nash dynamics that can occur under CGT-MAS on a processor.

System Dynamics Illustration: We now illustrate with the help of an example the autonomous Nash dynamics that occur under *CGT-MAS*. *Nash dynamics is a series of best response myopic moves made by agents converging towards Nash equilibrium.* Figure 5 shows simple Nash dynamics that can occur on a processor with four agents (cores) a_1, a_2, a_3 and a_4. It starts execution with two tasks that are assigned empty core-coalitions C_1 and C_2. Let the task associated with C_1 have higher throughput than the task belonging to C_2 and in equilibrium C_1 and C_2 should be assigned three agents and one agent, respectively.

Round 1 (Figure 5a): initially both C_1 and C_2 are empty and all agents are unbounded. All agents evaluate the benefits of joining either C_1 or C_2. All come to the same decision of joining C_1 associated with the higher throughput task. *Round 2* (Figure 5b): a_1 evaluates the possibility of joining C_2 on behalf of all agents in C_1 and concludes that it is better off joining C_2. *Round 3* (Figure 5c): the dynamics stop because equilibrium is reached as neither a_1 wants to move from C_2 nor a_2 wants to move from C_1. *Round 4* (Figure 5d): the task associated with core-coalition C_1 enters a new phase and its IPC decreases. New equilibrium should have two cores assigned to both core-coalitions. a_1 still does not want to move from C_2 but a_2 now decides to move to C_2 from C_1. *Round 5* (Figure 5e): the dynamics stop as system reattains equilibrium. *Round 6* (Figure 5f): another task with core-coalition C_3 enters the system or wakes up from hibernation and has the highest throughput. New equilibrium has one agent assigned to C_1 and C_2, with two agents assigned to C_3. Note that a_1 and a_3 move to C_3 in parallel from C_2 and C_1, respectively. *Round 7* (Figure 5g): dynamics come to halt again as an equilibrium is achieved.

Complexity: Each round of *CGT-MAS* requires $|A|$ agents to perform $O(|C|)$ utility calculations as described in Equation (2). In the worst-case, equilibrium can take up to $O(|C|)$ rounds (refer Theorem 1). In total, a maximum of $O(|A||C|^2)$ calculations are required to ensure stability. However, the processing overhead is distributed across all cores resulting in $O(|C|^2)$ worst-case calculations per-core.

The many-core performance scheduling problem can also be solved optimally using a scheduler based on DP. The DP scheduler has a centralized overhead of $O(|A||C|^2)$, which is difficult to parallelize [16]. Every time speedup or IPC of any task changes, the optimal schedule needs to be re-evaluated. In many-cores, where $|C| >> 1$, changes are nearly continuous, making an online DP scheduler impractical.

DP scheduler has a space overhead of $O(|A||C|)$, while *CGT-MAS* space overhead is only $O(1)$. Under *CGT-MAS*, at worst $|C|$ messages are broadcasted every round. Thus, in the worst-case $O(|C|^2)$ messages need to be transmitted every scheduling epoch. These messages can be transmitted with low overhead using Network on Chip (NoC) proposed for many-core architectures.

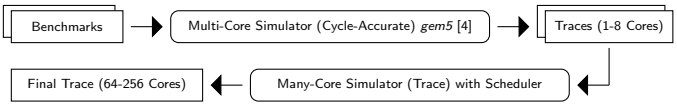


Figure 6: Experimental Setup.

3. EXPERIMENTAL EVALUATIONS

Experimental Setup: We use a two-stage adaptive many-core simulator for evaluation as shown in Figure 6. In the first stage, we use cycle-accurate *gem5* simulator [4] with up to eight cores with *ARMv7 ISA*. Each core is a 2-way out-of-order core with separate 64 KB L1 instruction- and data cache. All cores share a 2MB unified L2 cache. L1 caches are 4-way associative, while L2 cache is 8-way associative with all caches having a line size of 64 bytes. Multi-threaded tasks can run directly on the simulator. For sequential tasks, we modify the gem5 simulator to model *Bahurupi* adaptive multi-core architecture [14] that can execute a single sequential task on a virtual core-coalition of at most 8 cores. With increase in every core in the simulated system, cycle-accurate simulation time starts to increase exponentially, which makes cycle-accurate many-core simulations (with hundreds of cores) timewise infeasible. To bypass this limitation, we first collect isolated execution traces of the tasks from the cycle-accurate simulator, albeit restricted to core-coalition size of at most eight. We use a second trace-driven simulator that operates on these execution traces to model adaptive many-core with up to 256 cores. Akin to other trace-driven simulators, our simulations also cannot capture performance impacts due to resource contentions of concurrent tasks execution in many-core architectures [11].

We create workloads out of 26 sequential single-threaded (ILP) and 10 parallel multi-threaded (TLP) benchmarks as listed in Table 1. Sequential benchmarks comprise of integer-, float- and vision benchmarks from *SPEC* [9] and *SD-VBS* suites [18]. Parallel benchmarks come from *PARSEC* [3] and *SPLASH-2* [20] suites. Benchmarks are executed in syscall emulation mode. *SPEC-* and *SD-VBS-* benchmarks are executed with "ref"- and "full-hd' inputs, respectively. *PARSEC-* and *SPLASH-2* benchmarks are all executed with "sim-small" input.

All schedulers evaluated operate at a granularity of 10 million cycles. For a system running at 1GHz, this translates to decision making at 10ms, which is also the default operational granularity of *Linux* schedulers [12].

Table 1: List of all the benchmarks used in the evaluations.

Type	*Benchmark Name*
Integer	*astar, bzip2, gobmk, h264ref, hmmer, mcf, omnetpp, perlbench, sjeng, twolf, vortex*
Float	*art, bwaves, calculix, equake, gemsfdtd, lbm, namd, povray, tonto*
Vision	*disparity, mser, sift, svm, texture, tracking*
Parallel	*blackscholes, cholesky, fmm, fluidanimate, lu, radix, radiosity, swaptions, streamcluster, water-sp*

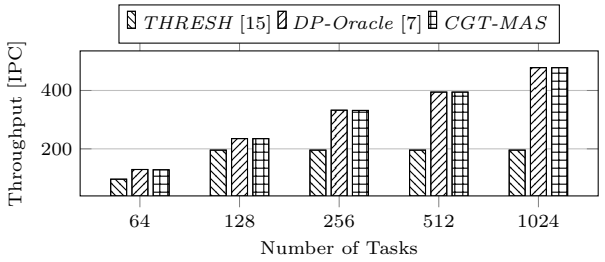


Figure 7: Performance of different schedulers on a closed 256-core many-core. Higher value of throughput is better.

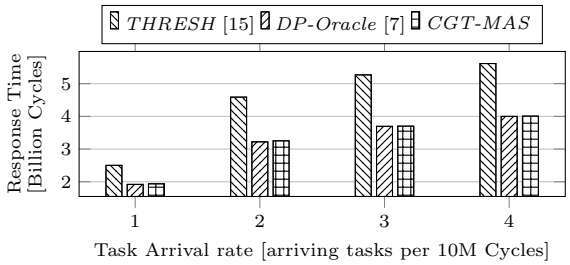


Figure 8: Performance of different scheduler on an open 256-core many-core. Lower value of response time is better.

Comparative Baselines: Authors in [7] presented a DP based centralized optimal scheduler for multi-cores called *Profile*, which takes in execution profiles of all tasks as an input and operates on average speedups. We extend this algorithm to *DP-Oracle*, which maximizes instantaneous IPC in every scheduling epoch to get our oracular comparison. Unlike *Profile* though, for a fair comparison *DP-Oracle* use IPC prediction instead of profiling; same as *CGT-MAS*.

$$\text{Maximize } \sum_{j=1}^{|C|} Q(C_j), \text{ given constraint } \sum_{j=1}^{|C|} |C_j| \leq |A|$$

where $Q(C_j)$ is the instantaneous IPC of the task associated with core-coalition C_j. For perspective, we also choose to compare against a threshold based heuristic called *Thresh* presented in [15]. *Thresh* assign cores to a task as long as task's gain in speedup from the core is more than 40%.

Closed System: We begin with a closed system on a 256-core many-core. In a closed system, the system begins with an immutable predefined set of tasks; instances of those tasks on completion immediately rejoin the system for re-execution. Throughput measured in terms of total system IPC is the standard performance metric for a closed system and highest possible value is desired. Figure 7 shows the throughput with different schedulers under full spectrum of system loads. For each experiment, ten random workloads are generated with uniform distribution among all benchmarks and the results obtained are then averaged across these 10 workloads. Figure 7 shows performance of *CGT-MAS* is equivalent to DP-Oracle under all loads. *Thresh* being heuristic in nature cannot optimally adapt to the workload and hence lags behind.

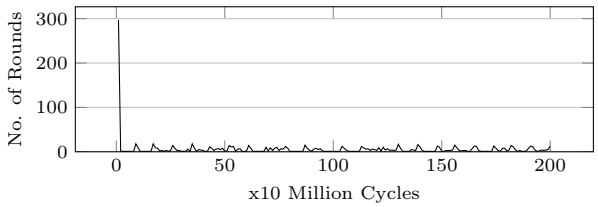


Figure 9: Number of MAS rounds occurring until convergence in each scheduling epoch under CGT-MAS on a 256-core many-core with 256 tasks.

Table 2: Overhead comparisons between *CGT-MAS* and *DP-Oracle* on varisized closed many-core systems.

Cores	Tasks	Total Processing Decrease	Per-Core Processing Decrease	Communication Increase
64	32	2.78x	160.85x	4.05x
	64	3.93x	228.46 x	2.84x
	128	2.84x	161.80x	3.67x
128	64	3.32x	380.94x	6.79x
	128	11.43x	1164.90x	2.26x
	256	5.96x	616.59x	3.77x
256	128	4.83x	1088.67x	9.88x
	256	8.31x	1724.49x	6.09x
	512	6.63x	1345.93x	6.8x

Open Systems: *CGT-MAS* is not limited to closed systems but can be applied to open systems as well. In an open system, tasks can arrive in the system at any time for execution and permanently leave once finished. Response time — measured as the time difference between task arrival and departure — is the standard performance metric for open systems. For each experiment, ten random workloads of 1024 tasks are generated with uniform distribution amongst all benchmarks and the results obtained are then averaged. The arrival time of the tasks follow Poisson distribution. Figure 8 shows the response time under different schedulers with different loads. Performance of *DP-Oracle* and *CGT-MAS* are also equivalent in open systems.

Convergence: *CGT-MAS* organizes itself autonomously to produce better results with every successive round of agent interactions until it converges. Figure 9 shows the number of rounds it takes for *CGT-MAS* to attain equilibrium in every scheduling epoch for a closed 256-core many-core system with full load (256 tasks). Convergence takes most number of rounds in the initial state because all agents are initially unbounded and system requires substantial reorganization. After that, system can operate efficiently with less rounds as only some of the tasks in the system change their behavior substantially at any given time.

Scalability: We use number of floating point operations performed and number of messages exchange by the agents in *CGT-MAS* as an approximate measure of its processing- and communication overhead, respectively. Total processing overhead is sum of all the floating point operations throughout the system in a scheduling epoch, while per-core processing overhead is the maximum operations performed by any core on behalf of its agents in that epoch.

In *DP-Oracle*, only one core does all the processing making the total processing overhead and per-core processing overhead equivalent. However, *DP-Oracle* has an advantage over *CGT-MAS* in terms of communication overhead as it requires only one round of communication in which all cores send their state information to one selected core that performs the scheduling. On the contrary, *CGT-MAS* requires several rounds of communications for the same. Table 2

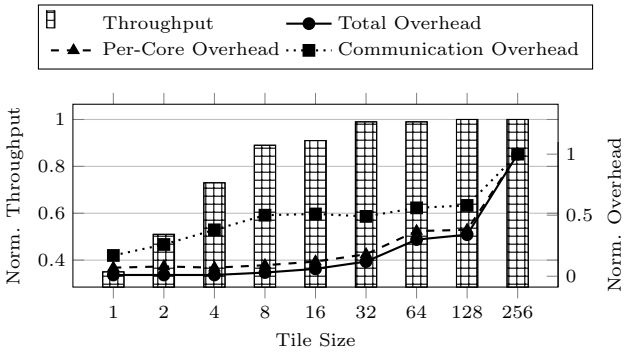


Figure 10: Normalized throughput and overheads under CGT-MAS on a half-loaded closed 256-core many-core where cores are segregated into varisized tiles.

shows 1000x reduction in per-core processing overhead attained by *CGT-MAS* over *DP-Oracle* at the cost of 10x enlargement of communication overhead when scheduling on varisized closed many-core systems under different loads.

Tiling: Due to performance penalty for maintaining cache coherency across spatially separated base cores, many-cores will be clustered into shared-cache tiles [8]. In tiled many-cores, only cores within a tile will be able to form core-coalitions amongst each other. Figure 10 shows the normalized throughput and overhead (against 256-core tile) under varisized tiles on a half-loaded closed 256-core many-core system. Figure 10 shows that keeping all cores in one tile (256-core tile) has no substantial performance advantage over a many-core in which cores are divided equally into two tiles (128-core tile). However, tiling can substantially reduce scheduling overheads. As long as tile size remains sufficiently larger than the average size of core-coalitions that could have been formed without the tiling restriction, the effects on throughput are insignificant. As tile size is reduced further, performance starts to drop significantly.

4. CONCLUSION

We propose a MAS scheduler called *CGT-MAS* for distributed dynamic scheduling on many-cores. *CGT-MAS* theoretically guarantees convergence to the optimal solution in a given number of steps from any initial state. Since *CGT-MAS* disburses its processing overhead across all the cores in a many-core, it can scale up as number of cores in the many-core increase. This scheduling can also be done optimally using a DP-based centralized scheduler called *DP-Oracle*. Our evaluations show that both schedulers reach the same solutions but *CGT-MAS* does so with 1000x reduction in per-core processing overhead. Thus, *CGT-MAS* is more suited for performing scheduling on many-core than *DP-Oracle* as it provides scalability without making any compromise on the quality of the solution obtained. Scheduling on multi-cores has been extensively explored but many-cores bring a new a paradigm shift in which existing works need to be revisited. *CGT-MAS* represents a step in that direction.

Acknowledgement

This work was supported in parts by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre "Invasive Computing" (SFB/TR 89) and in parts by Huawei International Pte. Ltd. research grant in Singapore.

5. REFERENCES

[1] I. Ahmad, S. Ranka, and S. U. Khan. Using Game Theory for Scheduling Tasks on Multi-core Processors for Simultaneous Optimization of Performance and Energy. In *International Symposium on Parallel and Distributed Processing (IPDPS)*, 2008.

[2] J. Augustine, N. Chen, E. Elkind, A. Fanelli, N. Gravin, and D. Shiryaev. Dynamics of Profit-Sharing Games. *Internet Mathematics*, 2013.

[3] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008.

[4] N. Binkert et al. The gem5 Simulator. In *SIGARCH Computer Architecture News*, 2011.

[5] K. Burns and B. Hasselblatt. The Sharkovsky Theorem: A Natural Direct Proof. *Mathematical Monthly*, 2011.

[6] T. Ebi, M. Faruque, and J. Henkel. TAPE: Thermal-Aware Agent-Based Power Economy Multi/Many-core Architectures. In *International Conference on Computer-Aided Design (ICCAD)*, 2009.

[7] D. P. Gulati, C. Kim, S. Sethumadhavan, S. W. Keckler, and D. Burger. Multitasking Workload Scheduling on Flexible-core Chip Multiprocessors. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008.

[8] J. Henkel, A. Herkersdorf, L. Bauer, T. Wild, M. Hübner, R. K. Pujari, A. Grudnitsky, J. Heisswolf, A. Zaib, B. Vogel, V. Lari, and S. Kobbe. Invasive Manycore Architectures. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2012.

[9] J. L. Henning. SPEC CPU2006 Benchmark Descriptions. *Computer Architecture News*, 2006.

[10] C. Kim, S. Sethumadhavan, M. S. Govindan, N. Ranganathan, D. Gulati, D. Burger, and S. W. Keckler. Composable Lightweight Processors. In *International Symposium on Microarchitecture (MICRO)*, 2007.

[11] K. M. Lepak, H. W. Cain, and M. H. Lipasti. Redeeming IPC as a Performance Metric for Multithreaded Programs. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2003.

[12] V. Pallipadi and A. Starikovskiy. The Ondemand Governor. In *The Linux Symposium*, 2006.

[13] A. Pathania, V. Venkataramani, M. Shafique, T. Mitra, and J. Henkel. Distributed Fair Scheduling for Many-Cores. In *Design, Automation and Test in Europe (DATE)*, 2016.

[14] M. Pricopi and T. Mitra. Bahurupi: A Polymorphic Heterogeneous Multi-core Architecture. *Transactions on Architecture and Code Optimization (TACO)*, 2012.

[15] M. Pricopi and T. Mitra. Task Scheduling on Adaptive Multi-Core. *Transactions on Computers (TC)*, 2013.

[16] A. Stivala, P. J. Stuckey, M. G. de la Banda, M. Hermenegildo, and A. Wirth. Lock-Free Parallel Dynamic Programming. *Parallel and Distributed Computing*, 2010.

[17] V. Vanchinathan. Performance Modeling of Adaptive Multi-core Architecture. Master's thesis, National University of Singapore, 2015.

[18] S. K. Venkata, I. Ahn, D. Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M. B. Taylor. SD-VBS: The San Diego Vision Benchmark Suite. In *International Symposium on Workload Characterization (IISWC)*, 2009.

[19] S. Wildermann, T. Ziermann, and J. Teich. Game-theoretic Analysis of Decentralized Core Allocation Schemes on Many-Core Systems. In *Design, Automation and Test in Europe (DATE)*, 2013.

[20] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Computer Architecture News*, 1995.