Q Learning Based Technique for Accelerated Computing on Multicore Processors

Avinash Dhole,
Scholor,CVRaman University,Kota,Bilaspur,Chattisgarh avi_dhole33@rediffmail.com

Dr Mohan Awasthy,
Professor, MPSTME Shirpur,Maharashtra mohanawasthy@yahoo.co.in

Dr Sanjay Kumar
Reader,SoS,Pt.Ravishankar Shukla University,Raipur,Chattisgarh sanraipur@rediffmail.com

Abstract: In this paper, we exhibit new convergent Q learning algorithm that consolidate components of policy iteration and classical Q learning/esteem iteration to effectively learn and control arrangements for a dynamic load adjusting situations utilizing reinforcement learning techniques. The model is prepared with a variation of memory optimization strategy for dynamic load adjusting recreation on multi-core processors making utilization of a machine learning approach, whose inputs areseveral time consuming computational processes and whose yield are time situated wrapper towards adjusting the computational and correspondence stack individually with an evaluation future rewards. The primary point of preference over this Q learning methodology is lower overhead; as most iteration doesn't require a minimization over all controls, in the context of modified policy iteration.We apply our technique to multi-core Q-learning way to make an algorithm which is a combination of the results from enhanced load and effective memory utilization on multiple cores. This technique gives a learning situation in handling computational load with no modification of the architecture resources or learning algorithm.These executions conquer a portion of the conventional convergence difficulties of offbeat modified policy iteration particularly in handling circumstances like that of multicore processors, and give policy iteration-like option Q-learning plans with as dependable convergence as classical Q learning.

Keywords: Multi-core Processing Reinforced Learning, Machine Learning, & Computational Load Balancing.

1. Introduction

With the coming of multi-core processors, the complexity of load balancing through scheduling threads has gone up impressively. Most schedulers take a gander at the need of threads that are prepared to rushed to settle on a scheduling decision. Since multi-core processors have shared resources e.g. the L2 cache, the conduct of a string running on one core can influence the performance of string running on different cores e.g. two threads which hoard the L2 cache if planned together on distinctive cores can have more awful performance than if they were co-booked with some different threads which did not hoard the cache [1-3]. The circumstance is more confused in light of the fact that the same string can change its conduct after some time e.g. it could be memory destined for quite a while and after that turn out to be computationally bound later on. It typically additionally is the situation that the threads correspond with one another to perform an assignment (this example likewise changes after some time) and it bodes well to co-plan these threads on diverse cores so they can share information through the L2 cache.

There is a requirement for a scheduler which can find out about which threads can be co-booked, which ought not be co-planned, which threads should be assembled together to keep running on the same core, remembering that the same string can carry on distinctively at diverse times. A scheduler with a static algorithm won't have the capacity to accomplish great performance in all situations.

Reinforcement Learning (RL) can be utilized here on the grounds that we have a method for offering input to the algorithm to say on the off chance that it is monitoring so as to looking so as to benefiting an occupation or not advance at the core use for every string e.g. planned (more the better), prepared (the less the better), obstructed (the less the better) and/or processor performance counters e.g. of guidelines resigned in an interim (the more the better), of cache misses (the less the better) , of synchronization directions fizzled (the less the better) [4-6]. These objectives can't be accomplished all in the meantime e.g. of synchronization disappointments can be brought around running all threads on the same core however this would prompt reduction in the total guidelines execution rate [7]. This issue calls for auto-tuning which can be accomplished by RL [8].

In [9], performance counters and address following office (not accessible on all processors) was utilized to discover cache sharing examples, threads were clustered taking into account these examples and after that threads in the same cluster were keep running on the same core. In [9], an investigative model of cache, memory progressive system and CPU was created which was then utilized alongside equipment performance counters to make determinations of when to move threads. In any of the above plans RL was not utilized nor did they display between procedure correspondences [10-12]. We proposes the utilization of machine learning innovation in outlining a self-streamlining, versatile memory controller equipped for arranging, learning, and ceaselessly adjusting to changing workload requests. We detail memory access scheduling as a support learning issue [14]. Reinforcement learning (RL) is a field of machine discovering that concentrates how self-sufficient specialists arranged in stochastic situations can learn ideal control approaches through cooperation with their surroundings. RL gives a general system to superior, self-upgrading controller outline. Theadvantagesof this method include massive memory capacity handling, the parameters of which were the equipment performance counters and different ones learnt utilizing RL [15, 16]. This capacity was keep running on every string to discover which string move would convey the most advantage to the entire multi-core processing framework.

2. Related Work

Chip Multiprocessors (CMPs) are alluring distinct options for solid cores because of their energy, performance, and complexity focal points [17, 18]. Current industry projections demonstrate that scaling CMPs to higher quantities of cores will be the essential instrument to profit from Moore's Law in the billion-transistor period. On the off chance that CMOS scaling keeps on taking after Moore's Law, CMPs could convey as much as double the quantity of cores and the accessible on-chip cache space at regular intervals [19]. Sadly, the advantages of Moore's Law are inaccessible to traditional bundling innovations, and hence, both the velocity and the quantity of flagging pins develop at a much slower rate (pin check increments by around 10% every year) [20]. Thus, off-chip transmission capacity might soon show a genuine obstruction to CMP versatility [21].

Yet giving sufficient top data transmission is just piece of the issue. Practically speaking, conveying a vast part of this hypothetical top to genuine workloads requests a memory controller that can successfully use the off-chip interface [22-25]. Measure scheduling is an intricate issue, requiring a sensitive harmony between bypassing access scheduling limitations, organizing asks for legitimately, and adjusting to a progressively changing memory reference stream [26]. At the point when stood up to with this test, existing memory controllers have a tendency to maintain just a little portion of the crest data transmission [27, 38]. The final result is either a huge performance hit, or an over-provisioned (and along these lines costly) memory framework [39-41].

Potential performance misfortune is relied upon because of access scheduling imperatives with a current DRAM controller. As the managed DRAM transfer speed of a case parallel application (SCALPARC [42]) with both a practical, contemporary controller outline (utilizing the FR-FCFS scheduling approach [43, 44]), and an idealistic (and unrealizable) plan that can support 100% of the controller's top data transmission, if enough request. With an idealistic scheduler, SCALPARC uses more than 85% of the memory transfer speed adequately. At the point when running with the sensible scheduler, in any case, the application's information transport use tumbles to 46% [45-47]. Henceforth, the effect of this scheduling wastefulness by and large L2 load misses punishment: while the idealistic scheduler accomplishes a normal burden miss punishment of 482 cycles, the practical scheduler encounters a 801-cycle normal punishment. The deciding result is a 55% performance misfortune contrasted with the hopeful scheduler [48].

A potential restricting component to higher performance in current memory controller outlines is that they are generally impromptu [49]. Commonly, a human picks a couple ascribes that are liable to be significant to upgrading a specific performance target (e.g., the normal holding up time of a solicitation in FR-FCFS) in light of related knowledge [50-54]. In view of these, the master devises an (altered) scheduling strategy fusing these traits, and assesses such an arrangement in a reenactment model. The subsequent controller as a rule needs two vital functionalities: First, it can't suspect the long haul outcomes of its scheduling decisions (i.e., it can't do long haul arranging). Second, it can't sum up and utilize the experience acquired through scheduling decisions made in the past to act effectively in new framework states (i.e., it can't learn) [55, 56]. As we will demonstrate, this inflexibility and absence of adaptivity can show itself as extreme performance corruption in numerous applications.

We propose to plan the memory controller as RL based thread scheduling operators whose objective is to learn naturally an ideal memory scheduling approach by means of communication with whatever remains of the framework. A RL-based memory controller takes as information parts of the framework state and considers the long haul performance effect of every move it can make [57, 58]. The controller's occupation is to (1) partner framework states and activities with long haul reward values, (2) make the move (i.e., plan the summon) that is evaluated to genius vide the most elevated long haul reward (i.e., performance) esteem at a given framework state, and (3) constantly overhaul long haul reward values connected with state-activity sets, in view of input from the framework, keeping in mind the end goal to adjust to changes in workloads and memory reference streams. Rather than ordinary memory controllers, a RL-based memory controller:

•Anticipates the long haul results of its scheduling decisions, and ceaselessly enhances its scheduling arrangement in light of this suspicion.

• Utilizes experience learned in past framework states to settle on great scheduling decisions in new, beforehand imperceptibly states.

•Adapts to progressively changing workload requests and memory reference streams.

A RL-based outline approach permits the equipment planner to concentrate on what performance focus on the controller ought to finish and what framework variables may be helpful to eventually infer a decent scheduling strategy, as opposed to contriving a settled arrangement that depicts precisely how the controller ought to achieve that objective [59]. This not just wipes out a great part of the human outline exertion included in conventional controller configuration, additionally (as our assessment appears) yields higher-performing controllers [60-63].

We assess our self-streamlining memory controller utilizing an assortment of parallelly executed software applications from the SPEC OpenMP, & NAS OpenMPbenchmark suites. On a 4-core CMP with a solitary channel DDR2-800 memory subsystem (6.4GB/s top data transmission in our setup), the RL-based memory controller enhances performance by 19% by and large (up to 33%) over a cutting edge FR-FCFS scheduler. This adequately slices down the middle the performance crevice between the single-channel arrangement and a more costly double channel DDR2-800 subsystem with double the crest data transmission. At the point when connected to the double channel subsystem, the RL-based scheduler conveys an extra 14% performance change by and large. By and large, our outcomes demonstrate that self-enhancing memory controllers can use the accessible memory data transfer capacity in a CMP all the more efficiently.

3. Background And Motivation

We briefly survey the operation of the memory controller in present day DRAM frameworks to inspire the requirement for wise threading and DRAM schedulers, and give foundation on reinforcement learning as pertinent to DRAM scheduling. Point by point depictions are past the extent of this paper. Intrigued perusers can discover more nitty gritty depictions in [64, 65] on DRAM frameworks and in [66, 67] on reinforcement learning.

3.1 Memory Controllers: Why it'sDifficult to Optimize?

Present day DRAM frameworks comprise of Dual In-Line Memory Modules (DIMMs), which are made out of multiple DRAM chips set up together to get a wide information interface. Every DRAM chip is composed as multiple free memory banks. Every bank is a two-dimensional cluster composed as lines × sections. Just a solitary column can be gotten to in every bank at any given time. Every bank contains a column cushion that stores the line that can be gotten to. To get to an area in a DRAM bank, the memory controller must first ensure that the line is in the column cradle. An actuate order brings the line whose location is demonstrated by the location transport from the memory cluster into the line cushion. Once the line is in the line support, the controller can issue read or compose orders to get to a section whose location is demonstrated by the location transport. Every read or compose summon exchanges multiple sections of information, indicated by a programmable burst length parameter. To get to an alternate column, the controller must first issue a precharge charge so that the information in the line support is composed back to the memory cluster. After the precharge, the controller can issue an enact order to open the new line it needs to get to.

The memory controller acknowledges cache misses and compose back solicitations from the processor(s) and supports them in a memory exchange line. The controller's capacity is to fulfill such demands by issuing suitable DRAM orders while saving the uprightness of the DRAM chips. To do as such, it tracks the condition of every DRAM bank (counting the line support), every DRAM transport, and every memory demand. The memory controller's undertaking is confused because of two noteworthy reasons.

To start with, the controller needs to comply with all DRAM timing constraints to give right usefulness. Measure chips have an extensive number of timing constraints that indicate when a summon can be lawfully issued. There are two sorts of timing constraints: neighborhood (per-bank) and worldwide (crosswise over banks because of shared resources between banks). An illustration nearby requirement is the initiate to peruse/compose delay, tRCD, which determines the base measure of time the controller needs to hold up to issue a read/compose order to a bank in the wake of issuing an actuate charge to that bank. A sample worldwide limitation is the keep in touch with read delay, tWTR, which determines the base measure of time that needs to go to issue a read summon to any bank in the wake of issuing a compose charge to any bank. Best in class DDR2 SDRAM chips frequently have a substantial number of timing constraints that must be obeyed when scheduling summons (e.g., more than 50 timing constraints in [68]).

Second, the controller must keenly organize DRAM summons from distinctive memory solicitations to advance framework performance. Diverse orderings and interleavings of DRAM charges result in distinctive levels of DRAM throughput and idleness. Discovering a decent timetable is not a simple assignment as scheduling

decisions have long haul outcomes: not just can issuing a DRAM order counteract adjusting different solicitations in consequent DRAM cycles (because of timing constraints), additionally the interleaving of solicitations from diverse cores (and in this way the future substance of the memory reference stream) is intensely affected by which core's blocking demands get overhauled next (conceivably unblocking the direction window or the bring unit of the asking for core, permitting it to create new memory demands). Additionally, a definitive advantage gave by a scheduling decision is vigorously affected by the future conduct of the processors, which positively is not under the scheduler's control. Current memory controllers utilize moderately basic strategies to calendar DRAM gets to. Rixner et al. [37] demonstrate that none of the altered approaches concentrated on give the best performance to all workloads and under all circumstances. On the other hand, the FR-FCFS (first-prepared first-start things out serve) arrangement [68, 69] gives the best normal performance. Among every single prepared charge, FR-FCFS organizes (1) section (CAS) orders (i.e., read or compose orders) over column (RAS) orders (i.e., initiate and precharge) with a specific end goal to boost the quantity of gets to open lines, and (2) more established summons over more youthful orders. Henceforth, FR-FCFS gives the most elevated need to the most established prepared section order in the exchange line. The objective of the FR-FCFS strategy is to expand DRAM throughput and minimize normal solicitation inactivity. Then again, FR-FCFS does not consider the long haul performance effect of either (1) organizing a segment order over a line charge, or (2) organizing a more established summoning over more recent command operations.

4. Methodology

Reinforcement Learning (RL) is a territory of the machine realizing which is worried with the connection of an agent with its surroundings. At every communication the agent detects the present conditions of nature, and picks an activity to execute. This activity causes changes in environment, in its turn, sends a scalar reinforcement signal r (penalty/reward) to the agent showing the adequacy of its activities. Along these lines, "The RL issue is intended to be a straightforward confining of the issue of gaining from collaboration to accomplish an objective". The RL issue can be settled by element programming and the ideal arrangement figured out whether the likelihood of prizes and state moves are known. In any case, this is not regularly the situation, and measurable testing routines were produced. One such approach is Q learning. In Q-learning agents figure out how to act ideally in a Markovian area by encountering the outcomes of their activities [70, 71]. An agent can use Q learning out how to secure an ideal strategy utilizing postponed rewards. The agent can locate the ideal strategy notwithstanding when there is no prior information of the impacts of its activities on the surroundings. Q learning uses the rewards and the best estimation of the present state to enhance the appraisal of the past state-activity pair. Now, we characterize a Markov Decision process (MDP) as takes after:

Definition 1 A Markov Decision Process MDP is a 4-tuple (S,A,T_p, R), where S is represented as a set of the states, A stand for set of actions, A(i) is the set of actions available at state.$T^{MDP}_{p(i,j)}(a)$ is the transition probability from state i to state j when performing action a ∈U(i) in state i, and $R_{MDP}(s, a)$ is the reward received when performing action a in state s.

We accept$R_{MDP}(s,a)$ as non-negative and restrictedby R_{Max}, i.e., $\forall s,a$: 0≤ $R_{MDP}(s,a) \leq R_{Max}$. For the ease of understanding we assume that the reward $R_{MDP}(s,a)$is deterministic, although all our results apply when $R_{MDP}(s,a)$is stochastic. A strategy for an MDP assigns, at each time *t*, for each state *s*a probability for performing action $a \in U(s)$, as per the given history as:

$$H_{t-1} = \{s_1, a_1, r_1, \dots, s_{t-1}, a_{t-1}, r_{t-1}\}$$

This incorporates the states, actions and rewards observed until time $t-1$. A policy P is memory-less technique, i.e., it primarily depends only upon the current state and not onto its history. Thus, a deterministic strategy P assigns each state a unique action.While taking after a strategy*P* we perform at time *t* action a_t at state s_t and observe a reward r_t (distributed according to $R_{MDP}(s,a)$*).* and the next state s_{t+1} (dispersed according to $P^{MDP}_{S_t,S_{t+1}}(a_t)$). We consolidate the sequences of rewards to a single value called the return, and our goal is to maximize it. Hence, we concentrateour work to focus on discounted return, which has a parameter $\gamma \in (0,1)$, and the discounted return of policy *P* is:

$$V^P_{MDP} = \sum_{t=0}^{\infty} \gamma^t r_t,$$

Where r_t is the reward observed at time *t*. Since all the rewards are bounded by $R_{Max}$the discounted return is limited by:

$$V_{Max} = \frac{R_{Max}}{1-\gamma}.$$

For a sequence of pairs for state and action, let the covering time, denoted by C', be an upper limit on the number of state-action pairs beginning from any pair, until all state-action appears in the sequential arrangement. Note that the covering time can be a function of both the MDP and the sequential arrangement or just of the sequence itself. At first we accept that from any beginning of a state, within C'steps all state-action pair sequence appear in the arrangement. Thereafter, we rest the assumption and accept that with probability at

least, from any start state in C'steps all state-action appears in the grouping. This underlying policy generates the sequence of state action pairs. The Parallel Sampling, P_S(MDP), returns for each pair (s,a) the next state s', dispersed according to $P_{S,S'}^{MDP}(a)$ and a reward r which is appropriated according to $R_{MDP}(s,a)$. The upsideof this model is that it permits to ignore the exploration and to concentrate on the learning. In some sense P_S(MDP) can be viewed as a flawless exploration approach. The Q-learning algorithm gauges the state-action value function (for discounted return) as takes after:

$$Q_{t+1}(s,a) = \alpha_t(s,a)\left(R_{MDP}(s,a) + \gamma \max_{b \in U(s')} Q_t(s',b)\right) + \left(1 - \alpha_t(s,a)\right)Q_t(s,a)$$

where*s'*is the state reached from state *s*when performing action *a*at time *t*. Since, Q-learning is anon-concurrentprocess as it updates a single entry every step. Thus, we now redesign the Q learning as a synchronous Q-learning, which performs the upgrades by using the parallel sampling $P_S(MDP)$. Our primary resultsare upper limits on the convergence rates and demonstrating their reliance on the learning rate. Forthe essential case is the synchronous Q-learning we need to demonstrate that for a polynomial learning rate we have a complexity, which is polynomial in $\frac{1}{1-\gamma}$. Interestingly, we demonstrate that linear learning rate has an exponential reliance on$\frac{1}{1-\gamma}$. Our outcomes display a sharp contrast between the two learning rates, in spite of the fact that they both merge with likelihood one. This refinement, which is highly important, can be watched just when we concentrate on the union rate, instead of joining in the breaking point. The limits for nonconcurrent Q-learning are comparable [72, 73]. The fundamental contrast is the presentation of a covering time*C'*. For polynomial learning rate we infer a bound polynomial in$\frac{1}{1-\gamma}$, and for linear learning rate our bound is exponential in$\frac{1}{1-\gamma}$. We likewise demonstrate a lower bound for linear learning rate, which is exponential. This suggests our upper limits are tight, and that the crevice between the two limits is genuine. We first give the outcomes for the synchronous Q-learning algorithm, where we overhaul every one of the sections of the Q capacity at every time step, i.e., the redesigns are synchronous.

Let $Q_T$be the value of the synchronous Q-learning algorithm using polynomial learning rate at time T. Then with probability at least 1-δ, we have that $\|Q_T - Q^*\| \leq \varepsilon$, given that

$$T = \Omega\left\{\left(\frac{V_{max}^2 \ln\left(\frac{|S||A|V_{max}}{\delta.\frac{1}{1-\gamma}.\varepsilon}\right)}{\left(\frac{1}{1-\gamma}\right)^2 \varepsilon^2}\right)^{\frac{1}{\omega}} + \left(\frac{\ln\left(\frac{V_{max}}{\varepsilon}\right)}{\left(\frac{1}{1-\gamma}\right)}\right)^{\frac{1}{1-\omega}}\right\}$$

The above bound is somewhat complicated. To simplify, assume that ω is a constant and consider first only its dependence on . This gives us linear time complexity for the synchronous learning rate.A parallel execution of a program is categorized into threesegmentssuch as:

(a) Parallelism phase,
(b) Computation phase and
(c) Computational Load scheduling phase.

Thus, the total completion time (C_T) for the execution of a program parallelly over multi-core processors can be represented mathematically in form of the following equation as:

$$C_T = C_P + C_{Co} + C_C$$

$$C_T = \left(c + j\sqrt{\log_2 n}\right)t_f + n.\theta.t_c + t_c(n)$$

Where, C_P is the completion time of executed operation for the division of task in paralleling sequence, C_{Co} is the completion time of executed operation for the scheduling between multiprocessors, C_C is the completion time of computation over all the processors. Also, c is the number of cycles, n is the number of processors, t_f is the average time to execute a flop by the processor, t_c is the time for the load balancing of the multiprocessors communication for the division and fetching of jobs, θ represents communication to communication ratio from multiple processing cores and t_c is the time taken for the computational operation over a processor.

Now, that we need to reduce the time taken for the parallelism of the jobs and communication overheads for the multi-core processing. Therefore, the objectives are respectively divided into two parts such as:

1. Synchronously Scheduling of Parallel Load Balancing: The computation load-balancing algorithm utilizes$C_P, C_{Co}, C_C$to balance theload. Each processor sends its processing load and Communication valuesto a central node every ccycles. The focal node pickstheprocessors which have the maximumworking loadand puts them off from scheduling by assigning it to $T_{p(i,j)}^{MDP}(a)$, which is the transition probability from state i to state j when performing action a . The lowest processors whichhave the minimum work load are put in action set A.

The processors of thesets A and $R_{MDP}(s,a)$ (which is the reward received when performing action a in state s) create a bipartite graph in which the weights estimations ofthe edges are the values of communication between multi processors. This implies that if Processor_1sent 1000 messages to Processor_2 since last execution of the dynamicload-balancing algorithm, there will be a link from P1 toP2 with a weight of 1000. Fundamentally, this graph demonstrates thecommunication history:

$$H_{t-1} = \{s_1, a_1, r_1, \dots, s_{t-1}, a_{t-1}, r_{t-1}\}$$

Thiscommunication history is of the top percentage of over-loaded and bottompercentage of under-loaded processors. We utilize a Graph BipartiteMatching Algorithm (GBMA)to match the load of processors of these twosets. After this matching, the central node informs the overloadedprocessors about their corresponding under-loadedprocessors with a Dynamic Destination Message (DDM). At whatever point processor P_i receives a DDM, itselects up to L number of Logical process (LPs) which have the most communication withthe destination processor, packs them into messages and thensends them to the destination processor.

2. Reduction of Computation to Communication Ratio: The correspondence load-adjusting algorithm has the same structure as the calculation load-adjusting algorithm. The primary distinction is that it endeavors to first adjust the correspondence and afterward the calculation. The algorithm utilizes 4-tuple MDP to adjust the load. Each c cycles, every processorPi sends its reduced return by$V_{Max} = \frac{R_{Max}}{1-\gamma}$values to a focal or central node. $\rho_{V,E}$contains thecommunication load between nodes *V*and communication lines corresponding to the edges *E*. The central nodefinds the maximum value of partition matrix of the logical processes $\rho_{V,E}$among all of thevalues that it received from different processors.If processors P_i and $P_j$had the most correspondence amid last c cycles the algorithm endeavors to exchange LPs between these two processors in order to put forththe verified results of the calculation, the processor with the highest estimation of the$\rho_{V,E}$ iis picked as the sender processor and a Dynamic Destination Message is sent to it. This procedure is preceded until rate of over-communicating and percentage of as under-communicating of the processors is coordinatedjointly.Upon the receptionof a DDM at processorP_i, it selects up to LPs which have the most communicationwith the endpointprocessor. These Logical Processes are sent to thedestination processor. As in the computation algorithm, if $P_i$gets a message which fits in with the LPs which were at that point exchanged, it forwards the message to their processors.

This is summed in the following algorithm:

Algorithm: ***Q-learning based Load Balancing Algorithm***

Input: N number of processors, communication load between nodes *V* and communication lines corresponding to the edges *E*, n number of processors, l=cost of synchronization, g=bandwidth

Output: *S*consists of scheduling table for communication step b/w multiple cores, computation steps of parallelizing and the synchronization step.

Step 1: Master Node (Core_0):

After each C cycles ← Update the state and actions using

$$H_{t-1} = \{s_1, a_1, r_1, \dots, s_{t-1}, a_{t-1}, r_{t-1}\}$$

Step 2: Evaluate thresholds of the partition matrix:

$$\sum_{i=1}^{cc} \rho_{V,E} = \begin{cases} 1 \; if \; g \geq \frac{\sum_{i=0}^{c} w}{\sum_{j=1}^{c'} l} \\ 0 \; if \; g > \frac{\sum_{i=0}^{c} w}{\sum_{j=1}^{c'} l} \end{cases}$$

where, V & E are the vertex and edge of the graph of the memory page, w is the local computation in process. Also, the c is the number of iteration in computation and c' in the number of communication overheads.

Step 3: Repeat steps 1-2 until all state sets are mapped.

Step 4: Initialize Q learning

$$Q_{t+1}(s,a) = \alpha_t(s,a)\left(R_{MDP}(s,a) + \gamma \max_{b \in U(s')} Q_t(s',b)\right) + \left(1 - \alpha_t(s,a)\right)Q_t(s,a)$$

Return V_{Max}

Step 5: Evaluate the page rank of the vertex for each cores based on weighed ordering of $Q_{t+1}(s,a)$:

$$\varphi_V = \frac{n-E}{|V|} + \sum_{i=1}^{cc} \rho_{V,E}\, \varphi_E + l * g$$

Step 6: Compute T for all cores:

$$T = \Omega\left\{\left(\frac{V_{max}^2 \ln\left(\frac{|S||A|V_{max}}{\delta.\frac{1}{1-\gamma}.\varepsilon}\right)}{\left(\frac{1}{1-\gamma}\right)^2 \varepsilon^2}\right)^{\frac{1}{\omega}} + \left(\frac{\ln\left(\frac{V_{max}}{\varepsilon}\right)}{\left(\frac{1}{1-\gamma}\right)}\right)^{\frac{1}{1-\omega}}\right\}$$

Step 7: *While* (Core_i ≤ i) // for checking into each processing cores.

{

if ($\varphi_V < V_{Max}$) then allocate memory for computation:

$$M = \sum_{i=1}^{cc} \rho_{V,E} * \varphi_V$$

else if ($\varphi_V > V_{Max}$) then allocate memory:

Repeat steps 4, 5 & 6.

else

Skip and do the normal simulation

Core_i ++;

Return ,

$$H_{t-1} = \{s_1, a_1, r_1, \ldots, s_{t-1}, a_{t-1}, r_{t-1}\}$$

}

Step 8: End Process.

This lessens the variation of the memory page and maps the parallelization of computational jobs at once. Also, it imitates the upgraded mapping of programming model over multi-core framework. Alternate routines for the most part has one segment for every vertex, except the proposed algorithm utilizes the single adjusting comparison for parallelization relying on the reasonable transfer speed in synchronous with the computational workload that to one cycle in light of the page rank equations. This diminishes the memory mapping and subsequently prioritizes the employments in view of page positioning. Regardless of the rearranging the algorithm shuns the hashing table for viable memory use concerning the measure of the occupations required for adjusting the work load.

5. Results & Discussion

Our test stage comprises of 32 double center, 64 bit Intel processors. Each of these processors has 8 Gigabytes of inside memory. At first, the load appropriation between this center of a processor is performed by the working framework. The processors are associated with one another by method for a 1 Gigabyte for every second Ethernet. We used Message Passing Interface (MPI) as the communication stage between processors. MPI gives a solid system to sending and accepting messages between diverse processors.

As to our decision of the load balancing strategy take note of that we had executed the dynamic load-balancing algorithms which satisfy the two targets on the double i.e, to deal with the computation and communication by modifying the memory and thread of the computational workload. The principle point of this algorithm was to dynamically adjust the computational load, while the communication algorithm attempted to adjust the communication load between the nodes. From our test results, we noticed that the computation and communication algorithms created distinctive results for an alternate number of processors. In both the computation and communication of multi-core setting, we make utilization of a parameter the percentage of participation of every nodes (or cores) which took an interest in the algorithm. We noticed that when we utilized a little number of processors (e.g. 2-6) in the computation algorithm we couldn't have a substantial quality for percentage participation of multiple cores in light of the fact that the more nodes which participation in the processing of an algorithm the more LPs are moved in every load-balancing cycle, along these lines expanding the communication overhead in a little network. In the event that this increment is more than the speed-up that we can accomplish as a result of load-balancing, the aggregate simulation time increments. For clear reasons, diverse estimations of C significantly affected the simulation. This is reflected back in the figure 1 beneath.

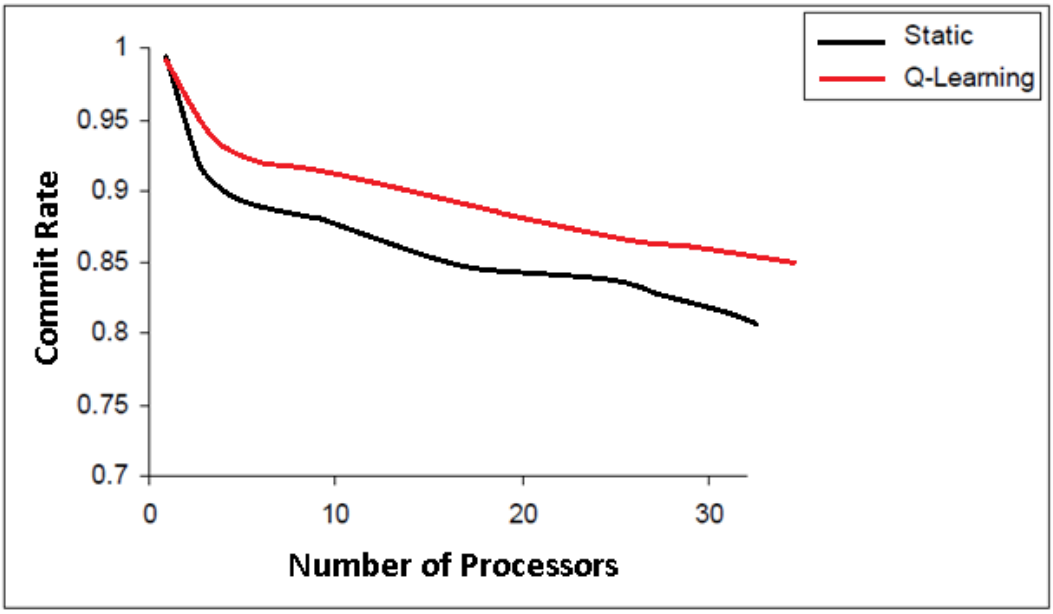

Figure 1: The average commit rate for different number of processor.

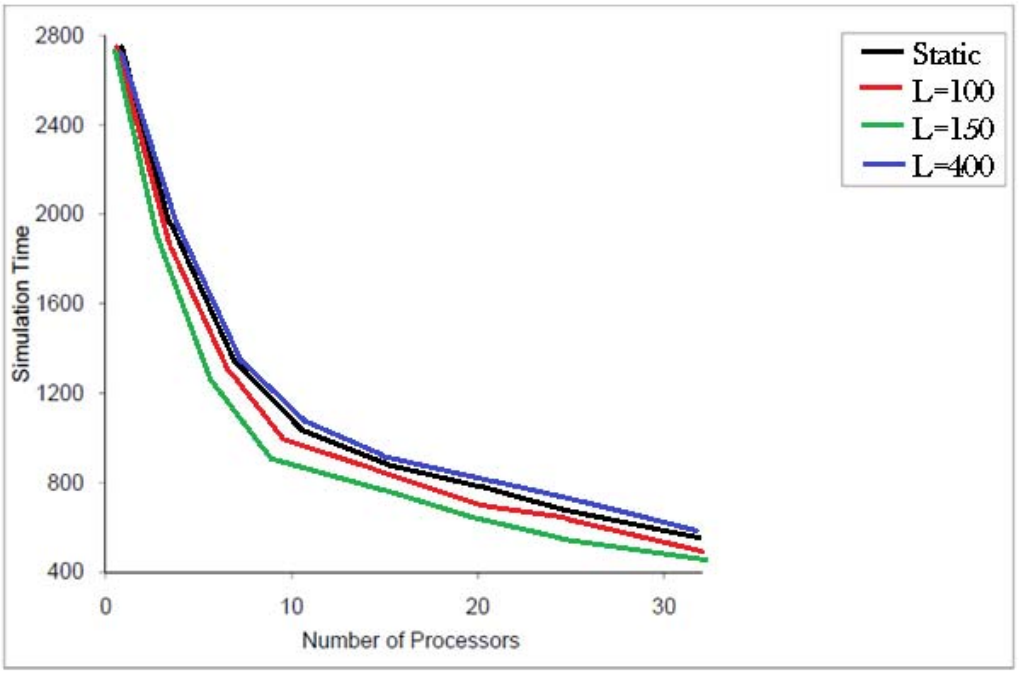

Figure 2: The average simulation time of the computation load balancing algorithm for different values of C and participation percentage of the processors at 50%.

Figure 2 demonstrates the execution of the computation Q-learning based dynamic load-balancing algorithm for unmistakable estimations of C when participation percentage of the processors at half on the OpenSparc T2 processor. The typical load qualification between most of the processors is decreased by up to 60% and we fulfilled up to a 18% change in the simulation time with C =150. As can be noted from figure 2, expanding the quantity of LPs from 100 to 150 results in better execution of the algorithm. Then again, expanding the quantity of LPs to 400 declines the circumstance. The purpose behind this is the point at which we move numerous LPs in each round, the communication time for exchanging the LPs increments and overpowers the execution pick up which we accomplished from balancing the load. The same result is additionally accomplished when participation percentage of the processors is changed to 20%. The simulation time of the OpenSparc T2 processor is up to 4% better with participation percentage of the processors at 10% than with 20%. The reason is that when we select more nodes to send LPs the communication time for exchanging the LPs increments. We don't put the aftereffects of the communication load balancing algorithm as a result of page farthest point.

Distinctive parameter values result in diverse simulation times for the communication load balancing algorithm too. We have additionally found that for an alternate number of processors and for distinctive circuits, we expected to use diverse load-balancing algorithms and their relating parameters to get the best execution. Hence, the proposed Q-learning based load balancing algorithm satisfies all the prerequisite without anyone else's input conforming with this changing circumstances and figures out how to set the parameters in like manner. Consequently, our significant goal for the Q-learning algorithm was to take in the sort of the dynamic load-balancing algorithm for an uncontrolled arrangement (diverse number of processors that take an interest in the load-balancing algorithm) and then to take in the relating parameters for that algorithm is effectively accomplished. Give us a chance to characterize the commit rate as the quantity of non-roll upheld messages isolated by the aggregate number of occasions. Figure 2 portrays the commit rate versus the quantity of processors with and without the Q-learning algorithm. The quantity of rollbacked messages increments with the quantity of processors. The purpose behind this is spreading out a greater amount of the LPs among the processors results in a more extended time for event cancellation. As can be seen, the learning algorithm has a superior commit rate than the simulation with static load-balancing for distinctive quantities of processors.

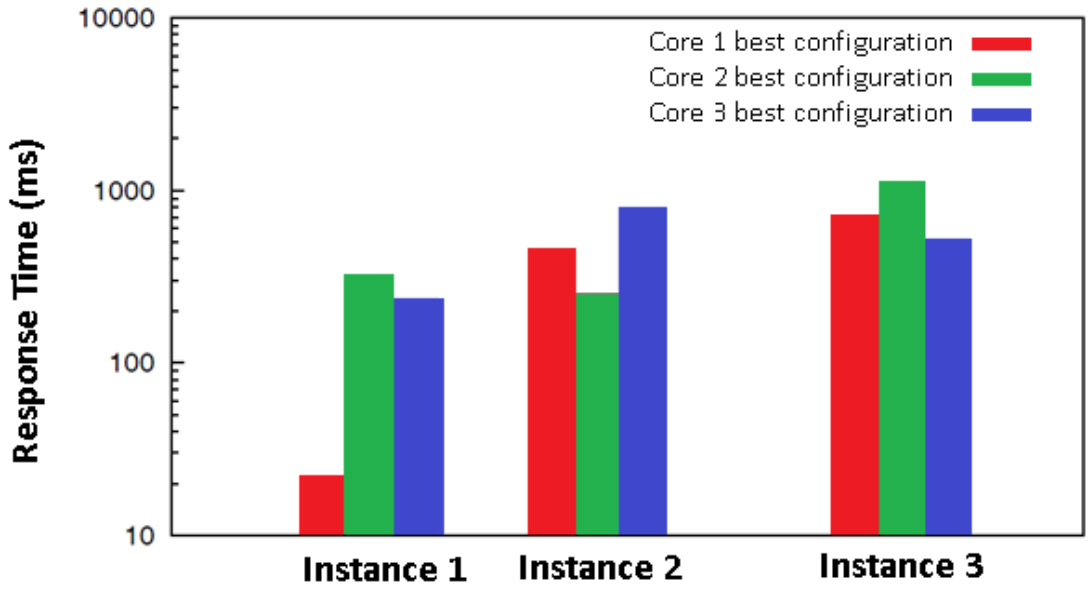

Figure 3: Performance under configurations tuned for different instances of workloads.

As can be found in figure 3, in the greater part of the cases the Q-learning enhances the simulation time more than different routines. In the event that the Q-learning algorithm improves result, its simulation time is at any rate in the same class as the best consequence of alternate algorithms. An intriguing point is the simulation time of the algorithms with two nodes. As can be seen, with less number of nodes the dynamic load-balancing algorithms can't enhance the simulation time as well as really intensifies the circumstance now and again. The explanation behind this is the point at which we have two nodes, the communication overhead of exchanging LPs is bigger than the advantage we accomplish from load-balancing. When we have more than four processors, the issue vanishes in the greater part of the cases and we can enhance the simulation time. Utilizing the Q-learning system, we can enhance the simulation time up to 89%, 89%, 87% and 85% for huge OpenSparc T2, LEON and RPI circuits separately.

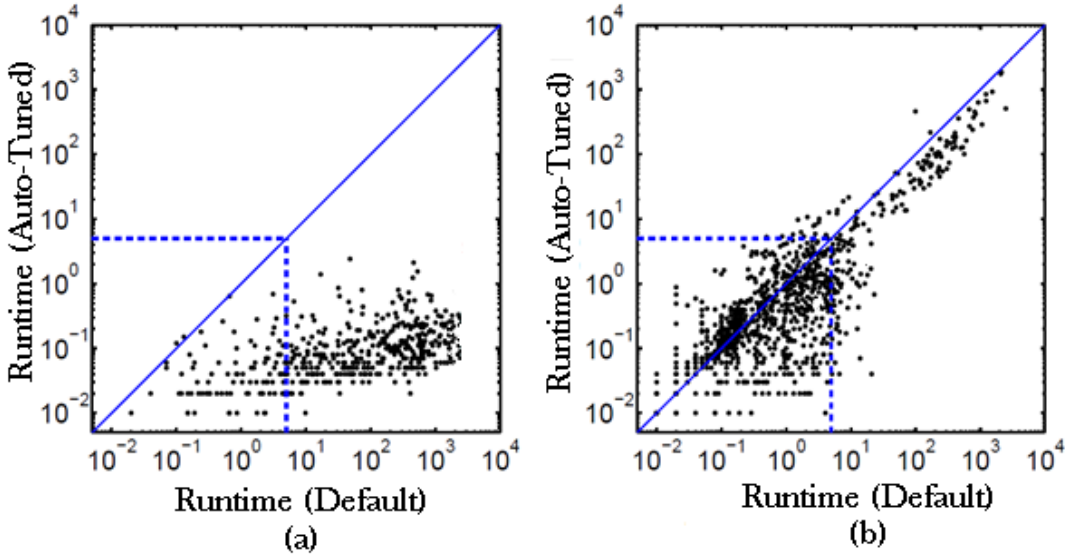

Figure 4: Correlation of default versus naturally decided parameter arrangements for (a) static situations (b) our proposed load balancing algorithm. Every dot represents one test instance. The dashed line at five CPU seconds demonstrates the cutoff time of the objective algorithm utilized amid the setup process.

In figure 4 above, we report the final performance accomplished by 30 autonomous keeps running of each configurator. For every independent configuration run, we utilized an alternate arrangement of training occurrences and seeds. We take note of that there was often a fairly vast difference in the performances found in distinctive keeps running of the configurators, and that the arrangement found in the keep running with the best training performance additionally tended to yield preferred test performance over the others. Hence, we utilized that setup as the consequence of algorithm arrangement. (Note that picking the arrangement found in the keep running with the best training set performance is a superbly honest to goodness technique since it doesn't require information of the test set. Obviously, the upgrades accordingly accomplished come at the cost of expanded general running time, however the free keeps running of the configurator can without much of a stretch be performed in parallel.). Figure 4 looks at the performance of this naturally discovered parameter design against the default arrangement, when runs are permitted to last up to 60 minutes. The speedups are more self-evident, since the penalized normal runtime in that table numbers runtimes bigger than five seconds as fifty seconds (ten times the cutoff of five seconds), while the information in the figure utilizes a much bigger cutoff time. The bigger speedups are most obvious as their relating speedup components in mean runtime are presently 270 separately.

6. Conclusion

In this paper we have considered the issue of learning based thread scheduling. We have displayed this new approach for planning string and memory controllers which work utilizing the standards of reinforcement learning (RL); also the scientific results so exhibited that bound the quantity of problematic moves made before touching base at a sub-optimal strategy with high sureness. A RL-based, self-advancing memory and string scheduling controller constantly and naturally adjusts its DRAM charge scheduling approach in view of its cooperation with the framework to streamline performance. Thus, it can use DRAM bandwidth more proficiently than a conventional controller that utilizes a settled scheduling arrangement. Our methodology likewise lessens the human outline exertion for the memory controller in light of the fact that the equipment fashioner does not have to devise a scheduling strategy that functions admirably under all circumstances. On a 4-core CMP with a solitary channel DDR2-800 memory subsystem (6.4GB/s crest bandwidth in our setup), the RL based memory controller enhances the performance of an arrangement of parallel applications by 34% overall (up to 67%) and DRAM bandwidth use by 19% by and large, over a cutting edge scheduler. Lament limits might make an interpretation of all the more promptly into insurances about transient constant performance impacts amid learning, as certifications with respect to cost (and henceforth esteem) decipher into assurances about assignment convenience. This change viably slices down the middle the performance crevice between the single-channel arrangement and a more costly double channel DDR2-800 subsystem with twice crest bandwidth. At the point when connected to the double channel subsystem, the RL-based scheduler conveys an extra 14% performance all things considered (up to 34%). We reason that RL-based self-advancing memory controllers give a promising approach to proficiently use the DRAM memory bandwidth accessible in a CMP.We have introduced exact results which propose that a learner that dependably utilizes its present data beats specialists which expressly empower investigation in this computational space.

References

[1] J. Carter et al. Impulse: Building a smarter memory controller. In HPCA-5, 1999.
[2] P. Auer, N. Cesa-Bianchi, and P. Fischer.Finite time analysis of the multiarmed bandit problem. Machine Learning, 47(2-3):235{256, 2002.
[3] P. Auer, T. Jaksch, and R. Ortner. Near-optimal regretbounds for reinforcement learning. In Advances in Neural Information Processing Systems, volume 21, pages 89{96, 2009.
[4] R. I. Brafman and M.Tennenholtz. R-MAX { a general polynomial time algorithm for near-optimal reinforcement learning. Journal of Machine Learning Research, 3:213{231, 2003.
[5] E. Brunskill, B. R. Leffler, L. Li, M. L. Littman, and N. Roy.Provably efficient learning with typed parametric models.Journal of Machine Learning Research, 10: 1955{1988, 2009.
[6] E. Even-Dar and Y. Mansour.Convergence of optimistic and incremental Q-learning. In Advances in Neural Information Processing Systems, volume 13, pages 1499{ 1506, 2001.
[7] E. Even-Dar, S. Mannor, and Y. Mansour. PAC bounds for multi-armed bandit and markov decision processes. In COLT '02: Proceedings of the 15th Annual Conference on Computational Learning Theory, pages 255{270, 2002.
[8] R. Givan, T. Dean, and M. Greig. Equivalence notions and model minimization in markov decision processes. Artificial Intelligence, 147(1-2):163{223, 2003.
[9] V. Aslot and R. Eigenmann.Quantitative performance analysis of the SPEC OMPM2001 benchmarks. Scientific Programming, 11(2):105–124, 2003.
[10] D. H. Bailey et al. NAS parallel benchmarks. Technical report, NASA Ames Research Center, March 1994.Tech. Rep. RNR-94-007.
[11] B. Calder, D. Grunwald, M. Jones, D. Lindsay, J. Martin, M. Mozer, and B. Zorn.Evidence-based static branch prediction using machine learning. ACM TOPLAS, 19(1), 1997.
[12] R. Caruana and D. Freitag. Greedy attribute selection. In International Conference on Machine Learning, pages 28–36, New Brunswick, NJ, July 1994.
[13] Y. Chang, T. Hoe, and L. Kaelbling.Mobilized ad-hoc networks: A reinforcement learning approach.In International Conference on Autonomic Computing, 2004.
[14] R. Sutton and A. Barto.Reinforcement Learning.MIT Press, Cambridge, MA, 1998.
[15] R. Glaubius, T. Tidwell, W. D. Smart, and C. Gill.Scheduling design and verification for open soft real-time systems. In RTSS'08: Proceedings of the 2008 Real-Time Systems Symposium, pages 505{514, 2008.
[16] R. Glaubius, T. Tidwell, C. Gill, and W. D. Smart.Scheduling policy design for autonomic systems. International Journal on Autonomous and Adaptive Communications Systems, 2(3):276{296, 2009.
[17] Advanced Micro Devices, Inc. AMD Athlon(TM) XP Processor Model 10 Data Sheet, Feb. 2003.
[18] J. H. Ahn, M. Erez, and W. J. Dally. The design space of data-parallel memory systems.In SC, 2006.
[19] Anandtech.IntelDeveloperForum 2007.http://www.anandtech.com/cpuchipsets/intel/showdoc.aspx?i=3102.
[20] ITRS. International Technology Roadmap for Semiconductors: 2005 Edition, Assembly and packaging. http://www.itrs.net/Links/2005ITRS/AP2005.pdf.
[21] L. Spracklen and S. G. Abraham. Chip multithreading: Opportunities and challenges. In HPCA-11, 2005.
[22] I. Hur and C. Lin. Adaptive history-based memory schedulers. In MICRO-37, 2004.
[23] D. A. Jimenez and C. Lin. Dynamic branch prediction with perceptrons. In HPCA-7, 2001.
[24] R. E. Kessler. The Alpha 21264 microprocessor. IEEE Micro, 9(2):24–36, Mar. 1999.
[25] A. McGovern and E. Moss.Scheduling straight line code using reinforcement learning and rollouts.In Advances in Neural Information Processing Systems, 1999.
[26] S. A. McKee et al. Dynamic access ordering for streamed computations. IEEE Transactions on Computers, 49(11):1255–1271, Nov. 2000.
[27] Micron. Technical Note TN-47-04: Calculating Memory SystemPower for DDR2, June 2006. http://download.micron.com/pdf/technotes/ddr2/TN4704.pdf.
[28] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens. Memory access scheduling.In ISCA-27, 2000.
[29] T. Dunigan, M. R. Fahey, J. White, and P. Worley.Early evaluation of the Cray X1.In SC, 2003.
[30] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: A 32-way multithreaded sparc processor. IEEE Micro, 25(2):21–29, 2005.
[31] J. D. McCalpin. Sustainable Memory Bandwidth in Current High Performance Computers. http://home.austin.rr.com/mccalpin/papers/bandwidth/.
[32] M. Joshi, G. Karypis, and V. Kumar.ScalParC: A new scalable and efficient parallel classification algorithm for mining large datasets. In IPPS, 1998.
[33] W. K. Zuravleff and T. Robinson. Controller for a synchronous DRAM that maximizes throughput by allowing memory requests and commands to be issued out of order. United States Patent #5,630,096, May 1995.
[34] O. Mutlu and T. Moscibroda.Stall-time fair memory access scheduling for chip multiprocessors.In MICRO-40, 2007.
[35] M. Ghosh and H.-H. S. Lee. Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3D Die-Stacked DRAMs. In MICRO-40, 2007.
[36] S. I. Hong, S. A. McKee, M. H. Salinas, R. H.Klenke, J. H. Aylor, and W. A. Wulf. Access order and effective bandwidth for streams on a direct Rambus memory.In HPCA-5, 1999.
[37] C. Natarajan, B. Christenson, and F. Briggs. A study of performance impact of memory controller features in multi-processor server environment. In WMPI, 2004.
[38] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith.Fair queuing memory systems.In MICRO-39, 2006.
[39] N. Rafique, W.-T.Lim, and M. Thottethodi. Effective management of DRAM bandwidth in multicore processors.In PACT, 2007.
[40] J. Shao and B. T. Davis.A burst scheduling access reordering mechanism.In HPCA-13, 2007.
[41] B. Sinharoy, R. N. Kalla, J. M. Tendler, R. J. Eickemeyer, and J. B. Joyner. POWER5 system microarchitecture. IBM Journal of Research and Development, 49(4/5):505–521, 2005.
[42] G. Tesauro et al. Online resource allocation using decompositionalreinforcement learning.In Conference for the American Association for Artificial Intelligence, July 2005.
[43] D. Vengerov and N. Iakovlev. A reinforcement learning framework for dynamic resource allocation: First results. In ICAC, 2005.
[44] Z. Zhang et al. A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality. In MICRO-33, 2000.
[45] Z. Zhu and Z. Zhang.A performance comparison of DRAM memory system optimizations for SMT processors.In HPCA-11, 2005.
[46] L. P. Kaelbling, M. Littman, and A. Moore. Reinforcement learning: A survey. Journal of Artificial Intelligence Research, 4:237{285, 1996.

[47] S. M. Kakade. On the Sample Complexity of Reinforcement Learning.PhD thesis, Gatsby Computational Neuroscience Unit, University College London, London, UK, 2003.
[48] S. M. Kakade, M. Kearns, and J. Langford.Exploration in metric state spaces. In ICML'03: Proceedings of the 20th International Conference on Machine Learning, pages 306{312, 2003.
[49] M. J. Kearns and S. P. Singh.Near-optimal reinforcement learning in polynomial time. Machine Learning, 2-3(49): 209{232, 2002.
[50] B. R. Leffler, M. L. Littman, and T. Edmunds.Efficient reinforcement learning with relocatable action models. In AAAI'07: Proceedings of the 22nd National Conference on Artificial Intelligence, pages 572{577, 2007.
[51] S. Mannor and J. N. Tsitsiklis.The sample complexity of exploration in the multi-armed bandit problem.Journal of Machine Learning Research, 5:623{648, 2004.
[52] V. Mnih, C. Szepesvari, and J.-Y.Audibert. Empirical bernstein stopping. In ICML '08: Proceedings of the 25th International Conference on Machine Learning, pages 672{679, 2008.
[53] V. Cuppu, B. Jacob, B. T. Davis, and T. Mudge.A performancecomparison of contemporary DRAM architectures.In ISCA-26, 1999.
[54] B. T. Davis. Modern DRAM Architectures.Ph.D. dissertation, Dept. of EECS, University of Michigan, Nov. 2000.
[55] D. Bertsekas. Neuro Dynamic Programming.Athena Scientific, Belmont, MA, 1996.
[56] T. Mitchell. Machine Learning.McGraw-Hill, Boston, MA, 1997.
[57] Micron. 512Mb DDR2 SDRAM Component Data Sheet: MT47H128M4B6-25, March 2006. http://download.micron.com/pdf/datasheets/dram/ddr2/512MbDDR2.pdf.
[58] S. Rixner. Memory controller optimizations for web servers.In MICRO-37, 2004.
[59] M. L. Puterman. Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley-Interscience, 1994.
[60] S. P. Singh and R. C. Yee.An upper bound on the loss from approximate optimal-value functions. Machine Learning, 16(3):227{233, 1994.
[61] A. Srinivasan and J. H. Anderson.Efficient scheduling of soft real-time applications on multiprocessors. Journal of Embedded Computing, 1(2):285{302, 2005.
[62] A. L. Strehl and M. L. Littman. An analysis of model based interval estimation for markov decision processes. Journal of Computer and System Sciences, 74(8):1309{ 1331, 2008.