Distributed Order Scheduling and its Application to Multi-Core DRAM Controllers

Thomas Moscibroda
Microsoft Research
Redmond, WA
moscitho@microsoft.com

Onur Mutlu
Microsoft Research
Redmond, WA
onur@microsoft.com

ABSTRACT

We study a distributed version of the order scheduling problem that arises when scheduling memory requests in shared DRAM systems of many-core architectures. In this problem, a set of n customer orders needs to be scheduled on multiple facilities. An order can consist of multiple requests, each of which has to be serviced on one designated facility, and an order is completed only when all its requests have been serviced. In the distributed setting, every facility has its own request buffer and must schedule the requests having only limited knowledge about the buffer state at other facilities.

In this paper, we quantify the trade-off between the amount of communication among different facilities and the quality of the resulting global solution. We show that without communication, the average completion time of all orders can be by a factor $\Omega(\sqrt{n})$ worse than in the optimal schedule. On the other hand, there exists a 2-approximation algorithm if the complete buffer states are exchanged in n communication rounds. We then prove a general upper bound that characterizes the region between these extreme points. Specifically, we devise a distributed scheduling algorithm that, for any k, achieves an approximation ratio of $O(k)$ in n/k communication rounds. Finally, we empirically test the performance of our different algorithms in a many-core environment using SPEC CPU2006 benchmarks as well as Windows desktop application traces.

Categories and Subject Descriptors

B.3.1 [**Memory Structures**]: Semiconductor Memories—*Dynamic memory (DRAM)*;
C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures;
F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms

Algorithms, Theory

Keywords

Distributed scheduling, order scheduling, distributed approximation, DRAM memory controllers, multi-core

PODC'08, August 18–21, 2008, Toronto, Ontario, Canada.

1. INTRODUCTION

In this paper, we study a distributed version of the so-called (customer) order scheduling problem (also referred to as the concurrent open shop scheduling problem) [8]. In this problem, a set of customer orders needs to be scheduled on multiple facilities. An order can consist of multiple requests (jobs), each of which has to be serviced on one particular facility. That is, unlike in general parallel machine scheduling problems, the facilities in our problem are dedicated, i.e., requests can only be serviced by one specific facility. The scheduler has to choose in which order the different requests are scheduled on the facilities. An order is completed when all its requests have been serviced and a natural objective function in many application scenarios is to minimize the total (or average) completion time of all orders.

The order scheduling problem has numerous practical applications, for instance in industrial manufacturing [3]. Essentially, the problem is applicable to any setting in which clients issue orders consisting of different parts, each of which has to be manufactured on a dedicated production resource. With this manufacturing background in mind, it is not surprising that the order scheduling problem has been studied exclusively in a centralized context (and in fact, the same is true for many other classic scheduling problems). Traditionally, it has been assumed that there is one central scheduler that controls access to all facilities and is therefore able to coordinate scheduling decisions across all facilities in a globally desirable way.

Motivated by a problem arising when scheduling memory requests in shared DRAM systems of multi-core computer architectures, we study the order scheduling problem in a distributed setting. In particular, we assume that facilities are distributed and that each facility has its own individual request buffer and scheduler that controls access to this facility. If the facilities are physically separated, then without communication (or any other means of sharing state information) each facility scheduler has knowledge only about the state of its own buffer and must base its scheduling decisions solely on this local information.

The problem is that such locally generated schedules can be globally suboptimal. To see this, consider a simple scenario in which there are two facilities, F_1 and F_2, and two orders O_1 and O_2. Order O_1 consists of two requests: R_{11} to facility F_1 and R_{12} to F_2. Order O_2 also consists of two requests, R_{21} and R_{22}, one to each facility. In a globally optimal solution, each of the two facilities would first schedule the corresponding requests of O_1, and subsequently the request of O_2. Assuming unit processing times for each request, orders O_1 and O_2 would be serviced at time 1 and 2, respectively, and the average completion time would be 1.5. In contrast, if both facility schedulers decide on their schedule locally, one facility may schedule R_{11} ahead of R_{21}, whereas the

other may schedule the requests in opposite order, i.e., R_{22} before R_{12}. As a result, both orders are finished only at time 2, i.e., the average completion time is 2. This simple example illustrates the trade-off between the amount of communication between facility schedulers and the quality of the resulting global solution. Intuitively, the more state information the facility schedulers exchange, the better they are able to coordinate their decisions, which can lead to shorter average completion times.

In this paper, we quantify this trade-off by providing new upper and lower bounds. In particular, we show that without communication, any distributed scheduling algorithm for the order scheduling problem may generate schedules that are by a factor $\Omega(\sqrt{n})$ worse than the optimal schedule, where n is the number of orders. In contrast, if schedulers can exchange their entire buffer state, factor 2 approximations become possible. We then prove a general upper bound on the achievable trade-off curve in between these two extremes points by studying a model in which each scheduler is allowed to broadcast state information for $\lfloor n/k \rfloor$ communication rounds, where $1 \leq k \leq n$ is an arbitrary parameter. We propose a distributed scheduling algorithm that, for any such k, achieves an approximation ratio of $O(k)$. We then empirically evaluate the performance of this algorithm compared to existing heuristics in a many-core environment using SPEC CPU2006 benchmarks as well as Windows desktop application traces.

2. MOTIVATION: REQUEST SCHEDULING IN SHARED DRAM MEMORY

Our impetus for studying order scheduling problems in a distributed context stems from our work on memory request scheduling in multi-core architectures. In such systems, multiple processing units (threads) on different cores share the same DRAM memory system. Modern DRAM chips are organized into different banks (=facilities), so that memory requests destined for different banks can be serviced in parallel. Each thread (=order) may simultaneously issue several requests, each of which must be serviced by a particular bank. Because, roughly speaking, a thread is stalled until all its outstanding memory requests are serviced by the DRAM[1], the goal is to minimize the average completion times of all threads that currently have outstanding memory requests. However, because each DRAM bank is controlled by an individual scheduler (the so-called bank-scheduler), the resulting order scheduling problem is of distributed nature.

While memory request scheduling in DRAM systems has served as our primary motivation to study this problem, there are many other settings in distributed databases or networks in which order scheduling problems arise in a distributed context. As we evaluate our methods using the framework of memory request scheduling, Section 7 provides additional relevant background about DRAM controllers in modern processing systems. A more detailed treatment of DRAM controllers can be found in [21, 14].

3. RELATED WORK

One of the key aspects characterizing our problem is that, unless there is a means for exchanging information about the state of the different buffers, each bank-scheduler (=facility) must take scheduling decisions based on local buffer information only. To the best of our knowledge, there are no studies on distributed order scheduling problems or, more specifically, on DRAM memory request scheduling problems in a distributed context.

Customer Order Scheduling: In a centralized context, the customer order scheduling problem[2] has been studied in several papers. The problem was proven to be NP-hard even for the case of three [8] and two [26, 22] facilities, respectively. In [8], a number of heuristics are discussed, all of which have worst-case approximation ratio of $\Omega(m)$, where m is the number of facilities. Based on an indexed linear programming formulation, a $16/3$-approximation algorithm for the weighted version was presented in [27]. Finally, as we discuss in Section 5, the work of [20] implies a 2-approximation algorithm although it does not explicitly state so. Several independent parties have then discovered this 2-approximation algorithm for the problem [2, 9, 5]. Whereas all of the above papers focus on minimizing the average (weighted) completion time of all orders, minimizing the number of tardy jobs has been studied in [17].

Memory Request Scheduling: Existing controllers for DRAM memory systems typically implement a so-called FR-FCFS scheduling algorithm [21] that does not require any coordination among bank schedulers. While the FR-FCFS scheduling policy optimizes the system throughput in single-core systems, it can be inefficient and unfair in many-core environments, and is even vulnerable to denial of memory service attacks [12]. Therefore, fairness-aware DRAM memory algorithms for multi-core systems have been proposed [14, 16, 15]. The batch-scheduling scheme discussed in Section 7 that forms the basis of our model has been proposed in [15]. It is currently the fairest and most efficient request scheduling algorithm for shared DRAM memory systems in many-core systems.

Finally, it is worth noting that there exist numerous other "distributed scheduling" problems that are unrelated to our work. In wireless networking, for instance, a distributed scheduling problem consists of finding time-slots for non-interfering transmissions using a distributed algorithm.

4. MODEL

Distributed Order Scheduling Problem: Let $\mathcal{T} = \{T_1, \ldots, T_n\}$ and $\mathcal{B} = \{B_1, \ldots, B_m\}$ denote the set of orders (threads) and facilities (banks) in the system. Each order T_i has a set R_{ij} of outstanding requests scheduled for B_j. The total processing time of all requests R_{ij} is denoted by p_{ij}. Let $R_j = \cup_i R_{ij}$ denote the buffer state of facility B_j, i.e., the set of all requests that need to be serviced by B_j. Each facility B_j is associated with an independent facility scheduler that controls access to this facility. Every facility B_j decides on the order in which its requests R_j are scheduled. Formally, a facility scheduler can therefore be considered a function that, based on R_j and all information obtained about the state of other buffers $R_k, k \neq j$, outputs an ordering $\omega_j = \langle T_1^j, T_2^j, \ldots, T_n^j \rangle$ over all orders.[3] Here, T_x^j denotes the order whose requests are scheduled at the xth position, after all requests from orders $T_1^j, \ldots, T_{x-1}^j$ have been fully processed by B_j. The totality of all local schedules ω_j then implies a global schedule ω. For a given schedule ω, we define C_{ij} to be the completion time of T_i on facility B_j, i.e., the earliest time when all requests R_{ij} have been serviced. The *completion time of a thread* T_i is $C_i = \max_{B_j \in \mathcal{B}} C_{ij}$. The objective function is to minimize the average completion time $\sum_{T_i \in \mathcal{T}} C_i / |T|$.

[1] In reality, the systems are very complex. A thread can make some, albeit little, progress even if one of its requests, rather than all, is serviced. However, previous work showed that for our purposes it is sufficiently accurate to assume that a thread is stalled until all its outstanding memory requests are serviced [6, 7, 13].

[2] The problem is sometimes also referred to as the *concurrent open shop* problem [22] as it is a relaxation of the classic open job shop problem in which—unlike in the original job shop problem—jobs can be processed in parallel by dedicated, request-specific facilities.

[3] It is known that there is an optimal schedule which is a *permutation schedule*, i.e., a schedule ω in which all orders are processed in the same order on all facilities, i.e., $\omega_j = \omega$ for all B_j [26].

Notice that in the context of the batch scheduling framework discussed in Section 7, we can assume that all outstanding requests are known at the outset of the algorithm, i.e., all requests have equal release time. Also, we can ignore requests that arrive later since such requests will be part of a subsequent batch.

If access to the different facilities is controlled by individual facility schedulers (as in the case of DRAM memory scheduling), the problem inherently becomes distributed. Hence, unless the entire state information R_j is exchanged between all facilities, it is generally difficult to ensure that 1) all facility schedulers output the same ordering ω_j and 2) that the resulting schedule is globally efficient. Intuitively, there exists a trade-off between the amount of communication between individual facility schedulers and the quality of the resulting schedule. Without communication, each facility scheduler must base its scheduling decision only on its own local information (i.e., the state of its own buffer), which can lead to globally suboptimal schedules. In order to formally capture this trade-off we propose the following distributed model.

Distributed Order Scheduling Model: Time is divided into synchronous communication rounds. Initially, each facility scheduler knows only about the state of its own local buffer. In each round, every facility scheduler can broadcast a message to all other facility schedulers. We assume that each message is of the form (T_i, p_{ij}), where T_i marks an order (=thread) and p_{ij} is the processing time of the order's requests to that facility. For simplicity, we only study algorithms that proceed along the following two-phases: 1) For some parameter k, facility schedulers exchange state information messages for $\lfloor n/k \rfloor$ rounds, and then 2) decide on the order in which the jobs/requests are scheduled. That is, no further communication takes place once the scheduling decisions are taken. The parameter k characterizes how much state information the facility schedulers can exchange before locally deciding on their scheduling order. If k is large, little communication is allowed, and if $k = 1$, the problem becomes equivalent to the standard non-distributed order scheduling problem since the schedulers can exchange their entire state information in $n = |T|$ rounds.

5. BASE CASES: NO COMMUNICATION VS. COMPLETE INFORMATION

Completely Local Scheduling Decisions: In this section, we establish results on the two base cases that are 1) no communication between the schedulers and 2) complete information exchange. In the former, every facility scheduler needs to decide on a thread ordering based entirely on the state of its local buffer R_j. In order to exclude any form of pre-determined scheduling based for instance on thread-IDs, we call a local facility scheduler *fair* if it decides on the ordering ω_j based only on the set of processing times of requests in its buffer, i.e., $\{p_{ij} | R_{ij} \in R_j\} \rightarrow \omega_j$. The following lower bound shows that in absence of communication and coordination between facility schedulers, the resulting global schedule may be highly suboptimal.

THEOREM 5.1. *Any (possibly randomized) fair distributed order scheduling algorithm in which schedulers do not communicate, has a worst-case approximation ratio of* $\Omega(\sqrt{|T|})$.

PROOF. Consider the following example consisting of n orders $T_1, \ldots, T_n$ and $m \leq n$ facilities $B_1, \ldots, B_m$. Let $\beta = n - m$. For each facility, there exists an order whose only request is destined for this facility. That is, for all such orders T_i, $i \in \{1, \ldots, m\}$, let $p_{ii} = 1$ and $p_{ik} = 0$ for all $k \neq i$. We call these orders *singleton orders* and their unique request singleton requests. For all orders T_i, $i \in \{m+1, \ldots, n\}$, let $p_{ik} = 1$ for all facilities $1 \leq k \leq m$.

The optimal global schedule first schedules on each facility B_j the singleton request from order T_j, followed by all the other requests from orders $T_{m+1}, \ldots, T_n$: $\omega_j^{OPT} = \langle T_j, T_{m+1}, \ldots, T_n \rangle$. The total completion time of this schedule is $OPT = \sum_i C_i^{OPT} \leq m + \beta \cdot \frac{\beta+2}{2}$. Since a fair facility scheduler cannot distinguish which among the $\beta + 1$ non-zero requests in its buffer is the singleton request, the best it can do is to schedule the requests in a random order. In expectation, the completion time of a singleton order is therefore $E[C_i] \geq \frac{\beta}{2}$. Hence, $E[ALG] = \sum_i C_i^{ALG} \geq \beta \cdot \frac{\beta+1}{2} + m \cdot \frac{\beta}{2}$. Substituting $m = n - \beta$, it follows that the approximation ratio α of any fair, local algorithm is at least $\alpha = \frac{E[ALG]}{OPT} \geq \frac{\beta \cdot (\beta+1) + m\beta}{2m + \beta(\beta+2)} = \frac{n\beta + \beta}{2n + \beta^2}$. This is minimized for $\beta = \sqrt{2n}$, which yields $\alpha > \frac{1}{2}\sqrt{2n} \in \Omega(\sqrt{n})$. □

It is interesting to note that most DRAM memory scheduling algorithms [21, 16] used in today's DRAM controllers belong to the category of fair and completely local algorithms captured in Theorem 5.1. That is, the total completion time of these scheduling policies can be by a factor $\Omega(\sqrt{n})$ worse than the optimal. As the number of cores on a chip (and with it n) is bound to increase in the future, this lower bound indicates the need for better coordination among bank schedulers in future DRAM memory systems.

Complete Information: In contrast, if the memory schedulers are capable of exchanging full state information among each other, significantly better solutions become possible. In particular, algorithms with an approximation ratio of 2 are known [2, 9, 5, 20].

The following linear program (denoted by OSLP) is a relaxation of the order scheduling problem.

$$\begin{aligned} \min \ & \frac{1}{|T|} \sum_{T_i \in \mathcal{T}} C_i \\ \text{s.t. } & C_i - C_{ij} \geq 0 && , \forall B_j \in \mathcal{B} \\ & \sum_{T_i \in X} p_{ij} C_{ij} \geq \frac{1}{2}\Big[\big(\sum_{T_i \in X} p_{ij}\big)^2 + \sum_{T_i \in X} p_{ij}^2 \Big] && , \forall X \subseteq \mathcal{T}, \forall B_j \in \mathcal{B} \end{aligned}$$

The first constraint describes that an order's completion time is the maximum over all facilities. The second set of constraints are relaxed versions of the *machine capacity constraints* first described by Wolsey [28] and Queyranne [19]. In particular, it is proven in [19] that on a single facility B_j, these linear inequalities completely describe the convex hull of feasible completion times vectors $(C_{1j}, \ldots, C_{nj})$. Furthermore, in spite of the exponential number of constraints, the constraint's separation problem and hence the LP itself can be computed in polynomial time. Intuitively, these constraints prevent too many requests from being completed too early. In [23, 20], these machine capacity constraints have been extended to parallel machine problems, similar to the formulation above.

The problem is that because these constraints are relaxations of the problem, there is generally no schedule that satisfies the completion times C_{ij} and C_i as computed by the LP. Consider for instance a facility B_j with two requests with processing time $p_{1j} = p_{2j} = 1$. While the LP can output $C_{1j} = C_{2j} = 1.5$, which satisfies the machine capacity constraints, there is clearly no real schedule in which both completion times are 1.5. We can use the following result by Schulz [23] in order to obtain an order-by-order bound on each facility:

LEMMA 5.2 ([23]). *Let $C_{1j}, \ldots, C_{nj}$ be a vector of completion times satisfying the machine capacity constraints, and assume w.l.o.g. that $C_{1j} \leq \cdots \leq C_{nj}$. Then, for each $i = 1, \ldots, n$, it holds that $\sum_{k=1}^{i} p_{kj} \leq 2C_i$.*

This lemma can be used to derive the following theorem on algorithms in which each local scheduler has complete state information. The proof consists only of putting together the above results and follows [20, 9].

THEOREM 5.3. *There exists a fair distributed order scheduling algorithm with communication complexity $|T|$ that achieves an approximation ratio of 2.*

PROOF. Let $\tilde{C}_i$ and $\tilde{C}_{ij}$ denote the completion times as computed by the LP. Because of the first constraints, it holds that $\tilde{C}_i \geq \tilde{C}_{ij}$ and hence, the vector $\tilde{C}_1, \ldots, \tilde{C}_n$ satisfies all LP constraints. Now, schedule all orders in $\mathcal{T}$ in non-decreasing order of $\tilde{C}_i$ and let C_i and C_{ij} denote the resulting actual completion times. It follows from Lemma 5.2 that for each order T_i and on each facility B_j, it holds that $C_{ij} = \sum_{k=1}^{i} p_{kj} \leq 2\tilde{C}_i$ and hence $C_i \leq 2\tilde{C}_i$. The theorem now follows from averaging over all T_i and the fact that $\sum_{T_i \in \mathcal{T}} \tilde{C}_i$ is a lower bound on the optimum solution. □

6. DISTRIBUTED SCHEDULING

In this section, we explore the trade-off between the amount of information exchange between the facilities and the achievable quality of the resulting global schedule. We propose and analyze a simple distributed algorithm that, for any parameter k such that $t := \lfloor \frac{n}{k} \rfloor \in \{0, \ldots, n-1\}$, has a running time of $t+1$ and achieves an approximation of $O(k)$.

6.1 Algorithm

The key idea of the algorithm is to prioritize the distribution of information about those requests at a given facility that can have the highest impact on the global scheduling decision. In contrast, information about requests that can have only little global impact are distributed in an aggregated fashion. In the absence of any a-priori global information, it is the "long" requests in a given facility B_j (requests with large processing time p_{ij} relative to other requests) that potentially have the highest impact on the global schedule. The reason is that an order is finished only when *all* its requests are serviced. Hence, if an order consists of one or more long requests in a facility, suboptimally scheduling the "short" requests of this order on the remaining facilities has no impact as long as they are not postponed too long.

The above intuition suggests that each facility in the distributed algorithm should broadcast information about its requests with highest processing times. If no additional information is exchanged, however, some critical piece of information is lost. In particular, facilities will have no knowledge about the *load*, i.e., the total processing time $\sum p_{ij}$, at the different facilities. Having knowledge about the other facilities' load is important. In the absence of such information, local facilities are unable to judge the relative importance of other facilities when deciding on their scheduling order. For an example, assume that there exists one facility on which every order has a very large request (relative to its requests on other facilities). In such a case, the optimal ordering of orders should simply follow the shortest-job-first principle, i.e., the scheduling order $\langle T_1, \ldots, T_n \rangle$ should correspond to a non-decreasing order of processing times on this facility. If, on the other hand, different orders have their large requests on different facilities, this simple strategy fails.

For the above reasons, our algorithm broadcasts exact information about the longest, most critical requests, and supplements this information with an *aggregate information* about all remaining requests, such that every facility is aware of its relative load. For convenience, define $t = \lfloor \frac{n}{k} \rfloor$ as the number of communication rounds minus 1. For a given facility B_j, we define the *long set* L_j to be the set consisting of the t orders T_i with the largest processing times p_{ij} for this facility. The *short set* S_j is the set of $n-t$ orders whose requests' processing times are not among the t longest for this facility. Notice that an order may be in the large set on some facilities, and in the short set on others.

Input: k
1: define $t = \lfloor \frac{n}{k} \rfloor$;
2: for each B_j, define $L_j = \{T_i \in \mathcal{T} \mid p_{ij}$ is among the t largest processing times for $B_j\}$; $S_j = \mathcal{T} \setminus L_j$;
3: **for each** $T_i \in L_j$ **broadcast** (T_i, p_{ij})
4: **broadcast** $(AVG, \overline{P}_j)$, where $\overline{P}_j = \frac{1}{n-t} \sum_{T_i \in S_j} p_{ij}$;
5: Locally invoke OSLP using for every facility $B_k \in \mathcal{B}$ the exact p_{ik} for all $T_i \in L_k$ and $\hat{p}_{ik} := \overline{P}_k$ for all $T_i \in S_k$.
6: Let C_i^{avg} be the resulting completion times from the above LP. Schedule the orders in non-decreasing order of C_i^{avg}.

Algorithm 1: DOS: Distributed Order Scheduling Approximation Algorithm (Code at Facility B_j)

Algorithm 1 proceeds as follows. In the first t communication rounds, every facility exchanges the processing times p_{ij} of all long orders $T_i \in L_j$. In the final $t+1$th round, each facility broadcasts the *average processing time* $\overline{P}_j$ of the remaining requests. Consequently, at the end of these $t+1$ rounds of broadcasts, every local facility scheduler knows the exact processing times of the long requests in each facility, as well as an average value of all remaining requests. Using this information, each local facility scheduler then locally invokes a version of the order scheduling linear program OSLP in Section 5, using the exact values p_{ij} for all long requests. For all other requests, the exact processing time is unknown and instead, the average value $\overline{P}_j$ from that facility is used as input to OSLP. The resulting completion times C_i^{avg} of this linear program are then sorted locally at each facility and the orders are scheduled in non-decreasing order of C_i^{avg}.

6.2 Analysis

The challenge when analyzing the performance of Algorithm 1 is to bound the suboptimality caused by the imprecision in the input of OSLP across different facilities. For instance, it may not be sufficient to show that on any single facility B_j, the sum of completion times, is not significantly increased due to the averaging of processing times of short requests. Because an order's completion time is the maximum over all facilities, such a proof does not prevent that the completion time of almost all orders increases, thereby causing a prohibitive increase of the sum of completion times.

As for notation, let LP_{ori} be the original OSLP linear program with the real processing times, and let LP_{avg} denote the averaged linear program used in Line 5 of Algorithm 1. C_i^{ori} and C_i^{avg} denote the optimal completion times of order T_i in LP_{ori} and LP_{avg}, respectively. Finally, C_i^{alg} is the actual completion time of T_i computed by the algorithm.

The analysis proceeds as follows, we first bound the gap between the optimal solution to LP_{avg} (as constructed by the algorithm) and the optimal solution to the original problem LP_{ori} by $O(k)$ (Lemmas 6.3 and 6.4). In the second step, we then show that the actual completion times resulting from scheduling the original requests (with processing times p_{ij}) based on the ordering computed using the averaged linear program is also within a factor $O(k)$ of the optimal solution. Finally, we show that our analysis is asymptotically tight by constructing a corresponding lower bound example for our algorithm.

We start with a simple helper lemma. In this lemma and its proof, we use the notational shortcut $j \in X_i$ to denote $(c_j, p_j) \in X_i$.

LEMMA 6.1. *Let $\mathcal{Q} = \{(c_1, p_1), \ldots, (c_r, p_r)\}$ be a set of pairs such that $p_i, c_i \geq 1$ for every $1 \leq i \leq r$. Further, let $\mathcal{X} = \{X_1, \ldots, X_s\}$ be a set of disjunct subsets of $\mathcal{Q}$ such that for every $X_i \in \mathcal{X}$, it holds that $\sum_{j \in X_i} p_j c_j \geq \frac{1}{2}\Big[\big(\sum_{j \in X_i} p_j\big)^2 + \sum_{j \in X_i} p_j^2\Big]$. It holds that*

$$\sum_{j \in \mathcal{X}} p_j c_j \geq \frac{1}{2s}\Big[\Big(\sum_{j \in \mathcal{X}} p_j\Big)^2 + \sum_{j \in \mathcal{X}} p_j^2\Big].$$

PROOF. It follows from the assumption in the lemma that the term $\sum_{j \in \mathcal{X}} p_j c_j$ can be bounded by

$$\begin{aligned}\sum_{j \in \mathcal{X}} p_j c_j \;&=\; \sum_{X_i \in \mathcal{X}} \sum_{j \in X_i} p_j c_j \;\geq\; \frac{1}{2}\left[\sum_{X_i \in \mathcal{X}} \Big(\sum_{j \in X_i} p_j\Big)^2 + \sum_{j \in \mathcal{X}} p_j^2\right] \\ &\geq\; \frac{1}{2}\left[\frac{1}{|\mathcal{X}|}\Big(\sum_{X_i \in \mathcal{X}} \sum_{j \in X_i} p_j\Big)^2 + \sum_{j \in \mathcal{X}} p_j^2\right],\end{aligned}$$

where the final inequality is due to $(\sum_i x_i)^2 / \sum_i x_i^2 \leq |x|$. The lemma now follows by replacing $\sum_{X_i \in \mathcal{X}} \sum_{j \in X_i} p_j$ with $\sum_{j \in \mathcal{X}} p_j$, and by pulling the term $\frac{1}{|\mathcal{X}|}$ in front of the parenthesis. □

We define Q_h to be the $t = \lfloor n/k \rfloor$ orders with highest completion time C_i^{ori}. The set Q_ℓ is the set containing the $n - t$ remaining orders with lower optimal completion times. We further define a value D as the average optimal completion time of all orders in Q_h, i.e., $D := \frac{1}{|Q_h|}\sum_{T_i \in Q_h} C_i^{ori}$. We can derive the following lower bound on D in terms of the aggregate values $\overline{P}_j$ at the different facilities.

LEMMA 6.2. *It holds that $D \;\geq\; \frac{n}{2}\left(1 - \frac{1}{k}\right) \cdot \max_{B_j \in \mathcal{B}} \overline{P}_j$.*

PROOF. Let $B_j \in \mathcal{B}$ be the facility with maximal $\overline{P}_j$. We show by contradiction that the claim holds for B_j. Assume for contradiction that $D < \frac{n}{2}\left(1 - \frac{1}{k}\right)\overline{P}_j$. Consider the set Q_ℓ of $n\left(1 - \frac{1}{k}\right)$ orders with lowest optimal completion time C_i^{ori}. By the definition of D, it holds for each order $T_i \in Q_\ell$ that $C_i^{ori} \leq D$. In the algorithm, the set S_j of orders, whose real processing time on B_j is unknown and replaced with $\hat{p}_{ij} = \overline{P}_j$ in LP_{avg}, consists of $n\left(1 - \frac{1}{k}\right)$ orders. Because these are the orders with *shortest* processing times in this facility, and because the cardinality of S_j is the same as Q_ℓ, we can observe that $\sum_{T_i \in Q_\ell} p_{ij} \geq \sum_{T_i \in S_j} p_{ij}$.

Based on the above inequalities, we now go on to show that if $D < \frac{n}{2}\left(1 - \frac{1}{k}\right)\overline{P}_j$, the OSLP constraint for set $Q_\ell \subseteq \mathcal{T}$ on facility B_j is violated. Specifically, the left hand side of this constraint is at most

$$\begin{aligned}\sum_{T_i \in Q_\ell} p_{ij} C_{ij}^{ori} \;&\leq\; D \cdot \sum_{T_i \in Q_\ell} p_{ij} \;<\; \frac{n}{2}\left(1 - \frac{1}{k}\right)\overline{P}_j \sum_{T_i \in Q_\ell} p_{ij} \\ &=\; \frac{1}{2}\sum_{T_i \in S_j} p_{ij} \sum_{T_i \in Q_\ell} p_{ij} \;\leq\; \frac{1}{2}\Big(\sum_{T_i \in Q_\ell} p_{ij}\Big)^2.\end{aligned}$$

In the above derivation, the equality follows from the fact that by definition $\overline{P}_j = \frac{1}{|S_j|}\sum_{T_i \in S_j} p_{ij}$ holds and hence, $\sum_{T_i \in S_j} p_{ij} = |S_j| \cdot \overline{P}_j = n\Big(1 - \frac{1}{k}\Big)\overline{P}_j$. All other inequalities follow from the discussion above.

The contradiction is now concluded by observing that the inequality $\sum_{T_i \in Q_\ell} p_{ij} < \frac{1}{2}\big(\sum_{T_i \in Q_\ell} p_{ij}\big)^2$ implies that the OSLP constraint for set Q_ℓ is violated. From this, the lemma follows. □

In the first step of the proof, we show that the optimal value of LP_{avg} is by at most a factor $O(k)$ larger than the optimal value of LP_{ori}. For this purpose, we define for each order $T_i \in \mathcal{T}$ a *virtual completion time* as $C_i^* := 2\max\{C_i^{ori}, 2D\}$.

LEMMA 6.3. *It holds that $\sum_{T_i \in \mathcal{T}} C_i^* \leq 2(2k+1)\sum_{T_i \in \mathcal{T}} C_i^{ori}$.*

PROOF. The sum of virtual completion times can be written as

$$\begin{aligned}\sum_{T_i \in \mathcal{T}} C_i^* \;&=\; 2\Big(\sum_{T_i | C_i^{ori} \geq 2D} C_i^{ori} + \sum_{T_i | C_i^{ori} < 2D} 2D\Big) \\ &\leq\; 2\Big(\sum_{T_i \in \mathcal{T}} C_i^{ori} + \frac{2 \cdot \big|T_i | C_i^{ori} < 2D\big|}{|Q_h|}\sum_{T_i \in Q_h} C_i^{ori}\Big).\end{aligned}$$

Because $\big|T_i | C_i^{ori} < 2D\big| \leq |\mathcal{T}|$ and $|Q_h| = |\mathcal{T}|/k$, it follows that $\sum_{T_i \in \mathcal{T}} C_i^* \leq 2(2k+1)\sum_{T_i \in \mathcal{T}} C_i^{ori}$. □

Having bounded by how much the virtual completion times can exceed the optimal completion times, we now need to show that the virtual completion times constitute a feasible solution to LP_{avg}.

LEMMA 6.4. *The set of virtual completion times C_i^* constitutes a feasible solution to LP_{avg}.*

PROOF. We prove the lemma by showing that if we set $C_{ij}^* := C_i^*$ in each facility B_j, the constraints of LP_{avg} are satisfied for every subset $X \subseteq \mathcal{T}$. Let $X \subseteq \mathcal{T}$ be an arbitrary such subset and consider the left-hand side of the corresponding OSLP constraint in LP_{avg}, $\sum_{T_i \in X} \hat{p}_{ij} C_i^*$, when using the virtual completion time. We rewrite this expression as $\sum_{T_i \in X} \hat{p}_{ij} C_i^* = \sum_{T_i \in X \cap L_j} \hat{p}_{ij} C_i^* + \sum_{T_i \in X \cap S_j} \hat{p}_{ij} C_i^*$ and study the two terms separately. For convenience, let $S_j^X = X \cap S_j$ and $L_j^X = X \cap L_j$. First, because the processing times p_{ij} of orders in L_j remain unchanged, $\hat{p}_{ij} = p_{ij}$, and because $C_i^* \geq 2C_i^{ori}$ we know that the virtual completion times of orders in L_j^X must satisfy the property

$$\begin{aligned}\sum_{T_i \in L_j^X} \hat{p}_{ij} C_i^* \;&\geq\; 2 \sum_{T_i \in L_j^X} \hat{p}_{ij} C_i^{ori} \\ &\geq\; 2 \cdot \frac{1}{2}\Big[\Big(\sum_{T_i \in L_j^X} \hat{p}_{ij}\Big)^2 + \sum_{T_i \in L_j^X} \hat{p}_{ij}^2\Big] \quad (1)\end{aligned}$$

since otherwise, the optimal completion times C_i^{ori} would be infeasible for the set $X \cap L_j$.

The more intricate case is the sum over all orders in $X \cap S_j$ because $\hat{p}_{ij}$ is no longer equivalent to p_{ij}, but instead, $\hat{p}_{ij} = \overline{P}_j$. We can lower bound the sum as

$$\begin{aligned}\sum_{T_i \in S_j^X} \hat{p}_{ij} C_i^* \;&=\; \overline{P}_j \sum_{T_i \in S_j^X} C_i^* \underset{(i)}{\geq} \overline{P}_j \cdot 4D \cdot |S_j^X| \\ &\underset{(ii)}{\geq}\; 2 \cdot \overline{P}_j^2 \cdot |S_j^X| \cdot |S_j| \underset{(iii)}{\geq} 2 \cdot \overline{P}_j^2 \cdot |S_j^X|^2 \\ &\geq\; 2 \cdot \frac{1}{2}\Big(|S_j^X| + |S_j^X|^2\Big)\overline{P}_j^2 \\ &=\; 2 \cdot \frac{1}{2}\left[|S_j^X| \cdot \overline{P}_j^2 + \Big(|S_j^X| \cdot \overline{P}_j\Big)^2\right] \\ &\underset{(iv)}{=}\; 2 \cdot \frac{1}{2}\left[\sum_{T_i \in S_j^X} \hat{p}_{ij} + \Big(\sum_{T_i \in S_j^X} \hat{p}_{ij}\Big)^2\right]. \quad (2)\end{aligned}$$

Inequality (i) is due to $C_i^* \geq 4D$. Inequality (ii) follows from Lemma 6.2. Inequality (iii) holds because S_j^X is a subset of S_j, and finally, Equality (iv) is true because $\hat{p}_{ij} = \overline{P}_j$ for all orders in S_j^X, and therefore $|S_j^X| \cdot \overline{P}_j = \sum_{T_i \in S_j^X} \hat{p}_{ij}$.

Inequalities 1 and 2 thus imply that for both subsets $X \cap L_j$ and $X \cap S_j$ of X, the OSLP constraint is satisfied with an extra "slack" factor of 2. We can now use Lemma 6.1 to show that the constraint is also satisfied for the entire subset X. Specifically, it follows from Lemma 6.1 (when identifying subsets $X \cap L_j$ and $X \cap S_j$ as subsets X_1 and X_2, respectively) that

$$\begin{aligned}\sum_{T_i \in X} \hat{p}_{ij} C_i^* &\geq 2 \cdot \frac{1}{4}\Big[\Big(\sum_{T_i \in X} \hat{p}_{ij}\Big)^2 + \sum_{T_i \in X} \hat{p}_{ij}^2\Big] \\ &= \frac{1}{2}\Big[\Big(\sum_{T_i \in X} \hat{p}_{ij}\Big)^2 + \sum_{T_i \in X} \hat{p}_{ij}^2\Big].\end{aligned}$$

Hence, all constraints in LP_{avg} are satisfied when using the virtual completion times C_i^*. □

Combining the two previous lemmas, we can conclude the first phase of the proof.

LEMMA 6.5. *It holds* $\sum_{T_i \in \mathcal{T}} C_i^{avg} \leq 2(2k+1) \sum_{T_i \in \mathcal{T}} C_i^{ori}$.

PROOF. Lemma 6.4 implies that the virtual completion times C_i^* form a feasible solution to LP_{avg} and therefore, $\sum_{T_i \in \mathcal{T}} C_i^* \geq \sum_{T_i \in \mathcal{T}} C_i^{avg}$. Finally, we can combine this with the bound derived in Lemma 6.3, $\sum_{T_i \in \mathcal{T}} C_i^* \leq 2(2k+1) \cdot \sum_{T_i \in \mathcal{T}} C_i^{ori}$. □

So far, we have shown that the optimal objective values of LP_{avg} and LP_{ori} differ by at most a factor of $O(k)$. However, we also need to show that when we actually schedule the original requests based on the ordering obtained after computing LP_{avg}, the resulting completion times C_i^{alg} are good.

For this purpose, we now define a new virtual completion time as $\widehat{C}_i := 2\max\{C_i^{avg}, 2D\}$. The difference between $\widehat{C}_i$ and the previously considered C_i^* is that unlike C_i^*, the values $\widehat{C}_i$ directly depend on C_i^{avg}, which will facilitate our reasoning about the algorithm's ordering.

LEMMA 6.6. *It holds* $\sum_{T_i \in \mathcal{T}} \widehat{C}_i \leq 2(6k+2) \cdot \sum_{T_i \in \mathcal{T}} C_i^{ori}$.

PROOF. Similar to the proof in Lemma 6.3, the sum of virtual completion times is

$$\begin{aligned}\sum_{T_i \in \mathcal{T}} \widehat{C}_i &= 2\Big(\sum_{T_i | C_i^{avg} \geq 2D} C_i^{avg} + \sum_{T_i | C_i^{avg} < 2D} 2D\Big) \\ &\leq 2\Big(\sum_{T_i \in \mathcal{T}} C_i^{avg} + \frac{2 \cdot |T_i | C_i^{avg} < 2D|}{|Q_h|} \sum_{T_i \in Q_h} C_i^{ori}\Big) \\ &\leq 2\Big(2(2k+1) \sum_{T_i \in \mathcal{T}} C_i^{ori} + 2k \sum_{T_i \in \mathcal{T}} C_i^{ori}\Big) \\ &= 2(6k+2) \sum_{T_i \in \mathcal{T}} C_i^{ori}.\end{aligned}$$

Where the last inequality follows from applying Lemma 6.5 (for the first term) as well as the transformation used in Lemma 6.3 (for the second term). □

LEMMA 6.7. *The virtual completion times $\widehat{C}_i$ form a feasible solution to LP_{ori}.*

PROOF. We show that when setting $\widehat{C}_{ij} := \widehat{C}_i$, the constraints of LP_{ori} are satisfied for every subset $X \subseteq \mathcal{T}$ and in every facility B_j. Again, we rewrite as $\sum_{T_i \in X} p_{ij} \widehat{C}_i = \sum_{T_i \in L_j^X} p_{ij} \widehat{C}_i + \sum_{T_i \in S_j^X} p_{ij} \widehat{C}_i$, and consider each of the two terms individually. By definition, it holds that $\sum_{T_i \in L_j^X} p_{ij} \widehat{C}_i \geq 2 \sum_{T_i \in L_j^X} p_{ij} C_i^{avg}$. As C_i^{avg} forms a feasible solution to the averaged linear program, and because for L_j^X it holds that $\hat{p}_{ij} = p_{ij}$, we have

$$\sum_{T_i \in L_j^X} p_{ij} \widehat{C}_i \geq 2 \cdot \frac{1}{2}\Big[\Big(\sum_{T_i \in L_j^X} p_{ij}\Big)^2 + \sum_{T_i \in L_j^X} p_{ij}^2\Big].$$

Now, consider the case of S_j^X in which generally, $p_{ij} \neq \hat{p}_{ij}$. We know from the definition of $\widehat{C}_i$ that $\widehat{C}_i \geq 4D$. Using this bound as well as Lemma 6.2, we can derive the following lower bound on $\sum_{T_i \in S_j^X} p_{ij} \widehat{C}_i$.

$$\begin{aligned}\sum_{T_i \in S_j^X} p_{ij} \widehat{C}_i &\geq 4D \cdot \sum_{T_i \in S_j^X} p_{ij} \\ &\geq_{\text{(Lemma 6.2)}} 2 \cdot \overline{P}_j \cdot |S_j| \cdot \sum_{T_i \in S_j^X} p_{ij} \\ &\geq 2 \cdot \Big(\sum_{T_i \in S_j^X} p_{ij}\Big)^2 \\ &\geq 2 \cdot \frac{1}{2}\Big[\sum_{T_i \in S_j^X} p_{ij} + \Big(\sum_{T_i \in S_j^X} p_{ij}\Big)^2\Big].\end{aligned}$$

As in the proof of Lemma 6.4, we can now combine these two lower bounds for L_j^X and S_j^X using Lemma 6.1. From this, it follows that

$$\sum_{T_i \in X} p_{ij} \widehat{C}_i \geq \frac{1}{2}\Big[\Big(\sum_{T_i \in X} p_{ij}\Big)^2 + \sum_{T_i \in X} p_{ij}^2\Big].$$

This shows that the set of $\widehat{C}_i$ satisfies the constraints of LP_{ori}. □

Using the previous lemmas, we can now prove the actual completion times C_i^{alg} resulting from Algorithm 1 are efficient compared to the virtual completion times $\widehat{C}_i$.

LEMMA 6.8. *It holds that* $\sum_{T_i \in \mathcal{T}} C_i^{alg} \leq 2 \cdot \sum_{T_i \in \mathcal{T}} \widehat{C}_i$.

PROOF. Assume w.l.o.g. that the T_i are named in non-decreasing order of the completion times computed in Line 5, $C_1^{avg} \leq C_2^{avg} \leq \ldots \leq C_n^{avg}$. Because every scheduler schedules the $T_i \in \mathcal{T}$ according to this order, it holds in every facility B_j that the completion time of T_i computed by the algorithm is $C_{ij}^{alg} = \sum_{k=1}^{i} p_{kj}$.

By Lemma 6.7, the set of $\widehat{C}_i$ is feasible for LP_{ori}. This implies that in each facility B_j, the constraints of OSLP are satisfied,

$$\sum_{k=1}^{i} p_{kj} \widehat{C}_k \geq \frac{1}{2}\Big[\Big(\sum_{k=1}^{i} p_{kj}\Big)^2 + \sum_{k=1}^{i} p_{kj}^2\Big]. \tag{3}$$

By the definition of the virtual completion times $\widehat{C}_i$, we know that if $C_a^{avg} \leq C_b^{avg}$ then $\widehat{C}_a \leq \widehat{C}_b$ also holds. It follows that $\widehat{C}_1 \leq \widehat{C}_2 \leq \ldots \leq \widehat{C}_n$, or alternatively $\widehat{C}_k \leq \widehat{C}_i$ for every $1 \leq k \leq i$. Therefore, $\sum_{k=1}^{i} p_{kj} \widehat{C}_k \leq \widehat{C}_i \sum_{k=1}^{i} p_{kj}$ and hence, we can rewrite Inequality (3) as

$$\widehat{C}_i \sum_{k=1}^{i} p_{kj} \geq \frac{1}{2}\Big[\Big(\sum_{k=1}^{i} p_{kj}\Big)^2 + \sum_{k=1}^{i} p_{kj}^2\Big].$$

When dividing both sides of the inequality by $\sum_{k=1}^{i} p_{kj}$, this implies that $\sum_{k=1}^{i} p_{kj} < 2 \cdot \widehat{C}_i$. The lemma now follows because for every $T_i \in \mathcal{T}$ and all facilities B_j, $C_{ij}^{alg} = \sum_{k=1}^{i} p_{kj}$. □

We now have all ingredients to prove the main theorem. It shows that the sum of completion times achieved by the algorithm can be at most by a factor of $O(k)$ larger than the optimal solution with global knowledge.

THEOREM 6.9. *Let OPT and $ALG(k)$ be the optimal solution with perfect global knowledge, and the solution achieved by Algorithm 1, respectively. It holds that $ALG(k) \leq 4(6k+2) \cdot OPT$.*

PROOF. Because LP_{ori} denotes the optimal fractional solution to the original problem, we know that its solution $\sum_{T_i \in \mathcal{T}} C_i^{ori}$ constitutes a lower bound on OPT. By Lemmas 6.6 and 6.8, we know that $\sum_{T_i \in \mathcal{T}} C_i^{alg} \leq 2 \sum_{T_i \in \mathcal{T}} \widehat{C}_i \leq 4(6k+2) \cdot \sum_{T_i \in \mathcal{T}} C_i^{ori}$, which proves the theorem. □

Tightness of Analysis: We now show that our analysis is asymptotically tight by presenting an example in which the schedule produced by Algorithm 1 is by a factor of $\Omega(k)$ worse than the optimal schedule. Intuitively, the proof consists of an example in which there are $2t$ orders having processing time 1 on every facility, while the remaining orders only have very short requests. Because the facility schedulers exchange information about only up to t orders, there remain t large orders that the facility schedulers do not have specific information about. Hence, instead of scheduling all short requests first, Algorithm 1 might schedule t large orders before all short ones, thereby unnecessarily delaying them.

THEOREM 6.10. *There are instances of the distributed order scheduling problem in which, for all k, the schedule produced by Algorithm 1 is by a factor of $\Omega(k)$ worse than the optimum.*

PROOF. Let $t = \lfloor n/k \rfloor$. In our example, the processing times of all orders $T_1, \ldots, T_t$ are $p_{ij} = 1$ on all facilities. The processing times of orders $T_{t+1}, \ldots, T_{2t}$ are $p_{ij} = 1 - \epsilon$ on all facilities, and all remaining processing times $T_{2t+1}, \ldots, T_n$ are $p_{ij} = \epsilon$ on all facilities B_j. In an optimal schedule, all orders are scheduled purely on a "shortest-job-first" basis, i.e., orders $T_{2t+1}, \ldots, T_n$ are scheduled first on all machines, followed by $T_{t+1}, \ldots, T_{2t}$ and finally $T_1, \ldots, T_t$. The sum of completion times in this schedule is no more than

$$OPT \leq \frac{\epsilon}{2}(n-2t)(n-2t+1) + 2t((n-2t)\epsilon + t + 1),$$

which, for $\epsilon \to 0$, approaches $OPT \leq 2t(t+1)$.

In Algorithm 1, all facilities broadcast the exact processing times of the $t = \lfloor n/k \rfloor$ requests $T_1, \ldots, T_t$ with largest processing times, but only average values for the remaining requests. Facility schedulers do not know the exact values of $T_{t+1}, \ldots, T_n$ and, hence, cannot distinguish between the long orders $T_{t+1}, \ldots, T_{2t}$ and the remaining short orders. For this reason, it is possible that the ordering computed in Line 6 first schedules orders $T_{t+1}, \ldots, T_{2t}$ before all short orders $T_{2t+1}, \ldots, T_n$. The sum of completion times resulting from this ordering is at least

$$ALG(k) \geq \frac{1}{2}t(t-\epsilon) + (n-2t)\Big(t(1-\epsilon) + \frac{1}{2}(n-2t)\epsilon\Big) + t\Big(t + \frac{t}{2}\Big).$$

For $\epsilon \to 0$, this approaches $ALG(k) \geq nt$. Hence, $\frac{ALG(k)}{OPT} = \frac{nt}{2t(t+1)} \geq \frac{kn}{2(n+k)} \in \Omega(k)$. □

7. BACKGROUND ON DRAM MEMORY AND DRAM CONTROLLERS

In this section, we describe how the distributed order scheduling problem models an important problem in shared DRAM memory scheduling in many-core systems.

Organization of DRAM memory and DRAM controller: As shown in Figure 1, the DRAM system in modern computer systems is organized into multiple *banks*, such that accesses to different banks can be serviced in parallel. Each core (i.e., processor or thread) connected to the DRAM can generate memory requests.

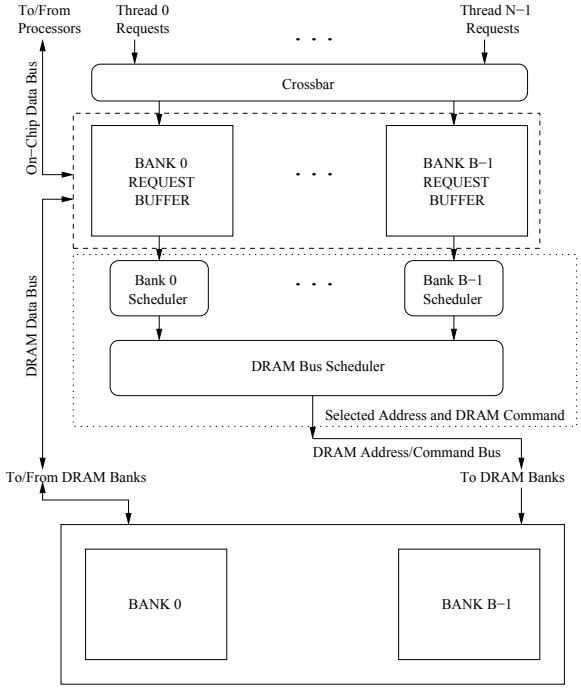

Figure 1: DRAM controller organization in modern multi-core processors

Each memory request is destined for a specific bank based on its address. To buffer outstanding requests, there is a *bank request buffer* associated with each bank. A *bank scheduler* operates on its local bank request buffer to determine which of the outstanding requests should be serviced next by that bank (if the bank is not already busy servicing a request). Due to packaging cost limitations, only one request can be sent to the DRAM at a given clock cycle, i.e. there is one single bus connected to each DRAM bank. Therefore, a separate *DRAM bus scheduler* chooses which bank scheduler's request will be serviced next. The DRAM bus scheduler usually takes the oldest request among the ones selected by the individual bank schedulers.[4] Note that a DRAM bank access takes hundreds of clock cycles; as such multiple requests can be serviced in parallel in DRAM banks. As a result, the local decisions made by each DRAM bank scheduler determines *which requests* are serviced in parallel in the banks, which is precisely the problem captured by our distributed order scheduling problem.

Minimizing the average completion time in our framework is the right objective, because at any given time, a thread can have multiple requests to different banks outstanding. Due to the nature of out-of-order instruction processing in modern processors, a thread is stalled until *all* of its outstanding memory requests are serviced [6, 7, 13]. Hence, as modeled by the distributed order scheduling problem, the execution time (i.e., completion time) of the thread will be determined by the bank that services the requests most slowly. For this reason, the decisions taken locally by each bank scheduler affect the completion time of a thread. And, the completion time of a thread is a critical measure to determine the scheduling efficiency in a DRAM controller. If the average completion time of all threads is low, the threads stall less and can make faster progress, ultimately leading to better performance.

[4]Note that this is true in the absence of any *row hits*, i.e. requests that hit in the row buffers associated with DRAM banks [12, 14]. Actual DRAM scheduling is significantly more complicated than what we describe. We only describe those scheduling decisions that result in first order performance effects to build our theoretical framework.

Batch-Scheduling: In order to avoid starvation and to guarantee efficient and fair distribution of the DRAM bandwidth to all cores sharing the DRAM system, *batch-scheduling of memory requests* has recently been introduced [15]. In this scheme, scheduling proceeds in batches. The idea of batching is to consecutively group outstanding requests in the bank request buffers into larger units called batches. Each bank marks the oldest N requests from each thread in its request buffer as belonging to the current batch. When scheduling, marked requests are prioritized over all other requests by the bank schedulers. Once no marked requests remain (i.e. all marked requests are serviced by the DRAM banks), a new batch is formed by repeating the marking process.

A thread's completion time within a batch is defined as the time between the initial formation of the batch and the time when the last request of the thread in the batch is serviced. As argued above, a thread's completion time within a batch determines its performance and in order to maximize overall system performance, a batch-scheduling based DRAM controller should schedule requests such that the average completion time of threads within a batch is minimized [15].

In view of the above, it is clear that the problem of scheduling DRAM memory requests in multi-core systems maps directly to the *order scheduling problem* outlined in the introduction. The banks correspond to the different facilities, and the threads correspond to orders. Within a batch, all requests issued by a certain thread to a certain bank can be regarded as one request.

8. EMPIRICAL EVALUATION

We evaluate the distributed order scheduling algorithm within the context of multi-core DRAM controllers, as described in Sections 2 and 7. We use microarchitectural simulation to empirically evaluate order scheduling and analyze its effects using real workloads. Our evaluation is based on the cycle-accurate simulation of a realistic multi-core system that implements the x86 instruction set architecture. The simulator takes as input instruction-level traces of x86 applications generated using the Pin [10] and iDNA [1] tracing tools. These instruction traces are then simulated via the processor models. Memory instructions, loads (reads) and stores (writes), access the processor's caches to load data. Each processor has a private L1 cache and a private L2 cache. A memory request that misses in both caches is entered into the corresponding bank request buffer in the DRAM controller. Each L2 cache is connected to the DRAM controller. Figure 2 shows the high-level architecture modeled by our simulator. We model especially the memory system in detail, faithfully capturing bandwidth limitations, contention, and enforcing bank/port/channel/bus conflicts. Table 1 shows the major DRAM and processor parameters.

8.1 Evaluated Applications

Table 2 describes the applications we have used in our evaluation. Table 3 then details the application mixes we have used to run on the different cores of the many-core system. Each application was compiled using gcc 4.1.2 with -O3 optimizations and run for 500 thousand instructions chosen from a representative execution phase using a methodology similar to [18].

Applications: We use several of the SPEC CPU2006 benchmarks [25], which are commonly used for processor performance evaluation, and two large Windows desktop applications (Matlab and an XML parsing application) for evaluation. We evaluate four different combinations of multiprogrammed workloads running on 4- and 8-core systems. The applications and application combinations listed in Tables 2 and 3 are selected to evaluate the average case behavior of different scheduling algorithms.

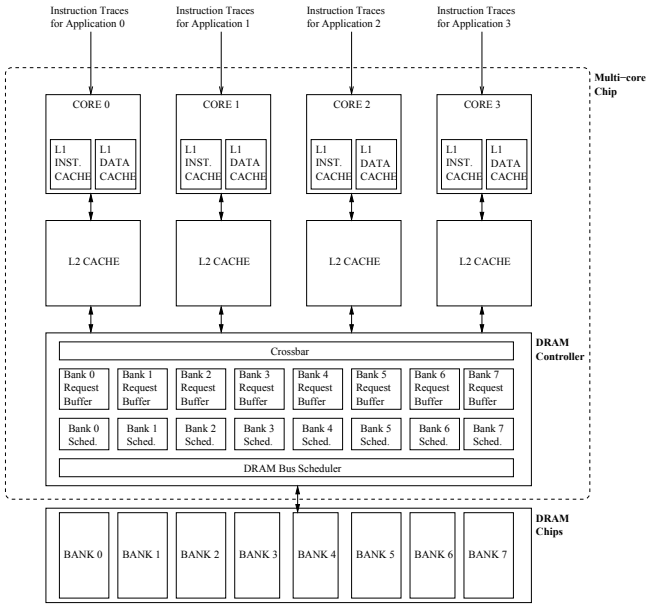

Figure 2: Simulated multi-core architecture

Metrics: We use the *average batch completion time (ABCT)* of threads to compare the scheduling efficiency of the controllers. A batch's average completion time is equal to the sum of completion times of the threads within the batch divided by the number of threads with marked requests. ABCT is computed by averaging average completion times over all batches at the end of the simulation runs. We also measure the system throughput provided by each controller, using the weighted-speedup metric, which is commonly used in multiprogrammed performance evaluation of microarchitecture designs [24].

8.2 Evaluated DRAM Scheduling Policies

We empirically evaluate several different schedulers, that use varying amount of communication between different bank (=facility) schedulers. All schedulers run within a batching scheme (see Section 7) to avoid starvation and ensure fairness [15]. The evaluated scheduling algorithms differ from each other in two aspects: 1) how they determine the *order* of threads to be serviced within a batch of requests, 2) how much information is communicated among the bank schedulers to compute the order of threads.

SJF scheduler: The *SJF* scheduler is the baseline scheduler if there is *no communication* between different bank schedulers. Each bank scheduler independently employs the *shortest job first* principle to decide the order in which it schedules its requests. As a result, the servicing order of threads in one bank can be completely different from the servicing order of threads in another bank.

Max-Total controller: The *MAX-TOT* scheduler [15] requires *complete thread information* among all bank schedulers. In particular, each bank scheduler conveys to every other bank scheduler the number of requests (in the current batch) from each thread in its own bank request buffer. Using this information, the schedulers compute the ordering of threads shown in Algorithm 2.

Since each bank scheduler has access to the same information, they all compute the same thread ordering, i.e., the servicing order of threads in all banks is the same. The MAX-TOT heuristic is based on the observation that the maximum number of outstanding requests to any bank correlates with the "shortness of the job," i.e., with the minimal memory latency required to serve all requests from a thread. A thread with smaller max-bank-load (MLB) has few

Cores and core pipeline	4 or 8 core systems; 4 GHz processor, 128-entry instruction window, 12-stage pipeline
Fetch/Exec/Commit width	3 instructions per cycle in each core; only 1 can be a memory operation
L1 Caches	32 K-byte per-core, 4-way set associative, 64-byte block size, 2-cycle latency
L2 Caches	512 K-byte per core, 8-way set associative, 64-byte block size, 12-cycle latency,
DRAM controller	128-entry request buffer per bank, reads prioritized over writes, XOR-based address-to-bank mapping [4]
DRAM chip parameters	8 banks; Micron DDR2-800 timing parameters (see [11]); 200-cycle bank access latency

Table 1: Baseline CMP and memory system configuration

Benchmark	Suite	Brief description
lbm	SPEC CPU2006 Floating-Point	Fluid dynamics; simulates incompressible fluids in 3D
mcf	SPEC CPU2006 Integer	Single-depot vehicle scheduling using combinatorial optimization
GemsFDTD	SPEC CPU2006 Floating-Point	Solves the Maxwell equations in 3D
omnetpp	SPEC CPU2006 Integer	Discrete event simulator modeling a large Ethernet campus network
matlab	Windows Desktop	Mathematical programming language and environment
leslie3d	SPEC CPU2006 Floating-Point	Computational fluid dynamics
libquantum	SPEC CPU2006 Integer	Simulates a quantum computer, running Shor's polynomial-time factorization algorithm
xml-parser	Windows Desktop	Parses and displays XML files
soplex	SPEC CPU2006 Floating-Point	Solves a linear program using a simplex algorithm and sparse linear algebra
cactusADM	SPEC CPU2006 Floating-Point	Solves the Einstein evolution equations
astar	SPEC CPU2006 Integer	Pathfinding algorithms for 2D maps
hmmer	SPEC CPU2006 Integer	Protein sequence analysis using profile hidden Markov models
h264ref	SPEC CPU2006 Integer	A reference implementation of H.264 video compression standard
gromacs	SPEC CPU2006 Floating-Point	Molecular dynamics; simulates Newtonian equations of motion
bzip2	SPEC CPU2006 Integer	In-memory compression/decompression of input files

Table 2: Evaluated applications

Combination	Applications
MIX1	lbm, mcf, GemsFDTD, omnetpp
MIX2	matlab, leslie3d, libquantum, mcf
MIX3	xml-parser, matlab, soplex, lbm
MIX8-1	mcf, xml-parser, cactusADM, astar, hmmer, h264ref, gromacs, bzip2

Table 3: Evaluated application combinations

1: **Max rule:** For each thread, let max-bank-load (MBL) be the maximum number of requests for any bank. A thread with a lower MBL is ordered before a thread with a higher MBL.
2: **Tie-breaker Total rule:** If two threads have the same MBL, a thread with lower total number of requests (in all banks) is ordered before a thread with higher total number of requests.

Algorithm 2: Max-Total Controller: Thread Ordering

marked requests going to the same bank and hence can be finished fast. By prioritizing requests from such threads and allowing banks to make coordinated thread ordering decisions, *MAX-TOT* aims to minimize the average completion time within a batch. It can be shown (using an example similar to the one used in the proof of Theorem 5.1) that MAX-TOT has a worst-case performance as bad as $\Omega(\sqrt{n})$. As our evaluations show, however, its performance is quite good in the practical cases.

Distributed Order Scheduling (DOS) Controller: This controller is the one described in Algorithm 1 of Section 6. The amount of information communicated between the schedulers varies depending on the parameter $t = \lfloor n/k \rfloor$. If $t = n$, all schedulers have complete global information, whereas if $t = 0$, each bank scheduler knows only the average processing time per thread in every bank request buffer.

8.3 Experimental Results

Figure 3 shows the average batch completion times of the different scheduling algorithms on the simulated 4-core system for three workloads. Several observations are in order:

- Having no communication between bank schedulers (i.e. *SJF* scheduling) results in consistently higher average batch completion times compared to having even the minimal amount of communication (i.e., even compared to *DOS* with $t = 0$). While the worst-case analysis in Theorem 5.1 implies a similar result for the worst-case, the empirical evaluation suggests that both *MAX-TOT* and *DOS* substantially outperform a purely local algorithm in scenarios using Windows desktop application traces as well.
- As the amount of communication between bank schedulers increases, the scheduling efficiency of *DOS* increases. This is demonstrated by the decreasing average batch completion times observed with increasing t value. Interestingly, the performance increase is very gradual, suggesting that every new piece of information can effectively be used to improve the computed schedule.
- The DOS algorithm with complete information exchange between bank schedulers ($t = 4$) provides better scheduling efficiency than MAX-TOT. The reductions in average batch completion time provided by DOS are respectively 4%, 5.1%, and 3.6% compared to MAX-TOT. This indicates that Algorithm 1 outperforms SJF and MAX-TOT not only in the worst case, but also in the average case.

We also note that the scheduling efficiency of DOS with t=3 and t=4 is the same because communicating the average processing time of a single request maintains complete information.

Comparison to LP lower bound: It is interesting to compare our results with the lower bound provided by the optimal solution to OSLP. We found that the average batch completion times as determined by OSLP for each mix is respectively 383, 547, and 539 cycles for the three workloads. This suggests that the DOS algorithm (with $t = 4$) is at most, respectively, 12.5%, 5.5%, and 11.3% worse than the optimal solution in the three workloads. Notice that the solution to OSLP only implies a (potentially loose) lower bound on the optimal schedule, and we assume that DOS is in fact much closer to the real optimum than these values.

Effect on System Throughput: Our evaluation results show that the reduction in average batch completion time has indeed an impact on the overall system throughput. Specifically, DOS (with $t = 4$) provides respectively 1.1%, 0.8%, and 0.9% improvement in system throughput over MAX-TOT. Similarly, it improves system throughput by 2.1%, 1.1%, and 1.4% compared to SJF. Also, as

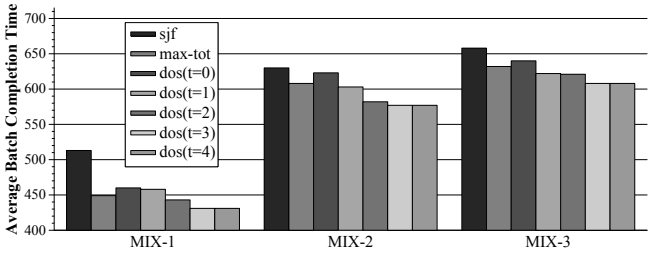

Figure 3: Average batch completion times (in processor clock cycles) of different scheduling algorithms in three different 4-core workloads

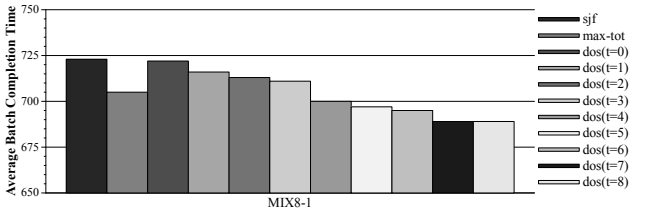

Figure 4: Average batch completion times of different scheduling algorithms in the 8-core workload

the information communicated between bank schedulers increases (from t=0 to t=4), system throughput also slightly increases.

8-Core Systems: Figure 4 shows the average batch completion times of the different scheduling algorithms on the simulated 8-core system. Note that average batch completion times are higher in the 8-core system than in the 4-core system because there is significantly higher pressure exerted on the DRAM system by 8 concurrently running applications. The conclusions from the 8-core system results are similar to the conclusions we have drawn from the 4-core system results. As a summary, we conclude that 1) the scheduling efficiency increases with more information exchanged among different bank schedulers, 2) having no communication among bank schedulers (SJF) results in the lowest scheduling efficiency (i.e. highest average batch completion time), and 3) distributed order scheduling with complete communication among bank schedulers provides the highest scheduling efficiency. In addition, in this average case, *DOS with* $t = 8$ achieves an average batch completion time that is at most 6.7% higher than the optimal solution as bounded from below by the solution to OSLP.

9. CONCLUSION

There has recently been a trend in the distributed computing community towards studying problems associated with multi- or many-core computing. So far, the problems most closely studied in this context deal with new programming paradigms such as transactional memory or parallel algorithms. In this paper, we have studied an important distributed computing problem that arises in the *microarchitecture* of multi-core systems. We feel that—following the same direction—there exist a vast number of important distributed computing problems in multi-core system architecture.

REFERENCES

[1] S. Bhansali et al. Framework for instruction-level tracing and analysis of programs. In *Proc. of the 2nd Conference on Virtual Execution Environments (VEE)*, 2006.

[2] Z. L. Chen and N. G. Hall. Supply Chain Scheduling: Assembly Systems. *Working Paper, Department of Systems Engineering, University of Pennsylvania*, 2000.

[3] I. Duenyas. Estimating the Throughput of Cyclic . *Management Science*, 39:616–625, 1993.

[4] J. M. Frailong, W. Jalby, and J. Lenfant. "XOR-Schemes: A flexible data organization in parallel memories". In *Proc. of International Conference on Parallel Processing (ICPP)*, 1985.

[5] N. Garg, A. Kumar, and V. Pandit. Order Scheduling Models : Hardness and Algorithms. In *Proc. of the Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2007.

[6] T. Karkhanis and J. E. Smith. A day in the life of a data cache miss. In *Workshop on Memory Performance Issues*, 2002.

[7] T. Karkhanis and J. E. Smith. A First-Order Superscalar Processor Model. In *Proc. of the 31th ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2004.

[8] J. Leung, H. Li, and M. Pinedo. Order Scheduling in an Environment with Dedicated Resources in Parallel. *Journal of Scheduling*, 8:355–386, 2005.

[9] J. Leung, H. Li, and M. Pinedo. Scheduling Orders for Multiple Product Types to Minimize Total Weighted Completion Time. *Discrete Applied Mathematics*, 155:945–970, 2007.

[10] C.-K. Luk et al. Pin: Building customized program analysis tools with dynamic instrumentation. In *Proc. of the ACM Conference on Programming Language Design and Implementation (PLDI)*, 2005.

[11] Micron. *1Gb DDR2 SDRAM Component: MT47H128M8HQ-25*, May 2007.

[12] T. Moscibroda and O. Mutlu. Memory Performance Hogs: Denial of Memory Service in Multi-Core Systems. In *Proc. of the 16th USENIX Security Symposium*, 2007.

[13] O. Mutlu, H. Kim, and Y. N. Patt. Efficient runahead execution: Power-efficient memory latency tolerance. *IEEE Micro: Top Picks from Computer Architecture Conferences*, 26(1):10–20, 2006.

[14] O. Mutlu and T. Moscibroda. Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors. In *Proc. of the 40th ACM/IEEE Symposium on Microarchitecture (MICRO)*, 2007.

[15] O. Mutlu and T. Moscibroda. Enhancing the Performance and Fairness of Shared DRAM Systems with Parallelism-Aware Batch Scheduling. In *Proc. of the 35th ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2008.

[16] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith. Fair Queuing Memory Systems. In *Proc. of the 39rd ACM/IEEE Symposium on Microarchitecture (MICRO)*, 2006.

[17] C. T. Ng, T. C. E. Cheng, and J. J. Yuan. Concurrent Open Shop Scheduling to Minimize the Weighted Number of Tardy Jobs. *Journal of Scheduling*, 6(4):405–412, 2003.

[18] H. Patil et al. Pinpointing Representative Portions of Large Intel Itanium Programs with Dynamic Instrumentation. In *Proc. of the 37rd ACM/IEEE Symposium on Microarchitecture (MICRO)*, 2004.

[19] M. Queyranne. Structure of a Simple Scheduling Polyhedron. *Mathematical Programming*, 58:263–285, 1993.

[20] M. Queyranne and M. Sviridenko. New and Improved Algorithms for Minsum Shop Scheduling. In *Proc. of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 871–878, 2000.

[21] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens. Memory access scheduling. In *Proc. of the 27th ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2000.

[22] T. A. Roemer. A Note on the Complexity of the Concurrent Open Shop Problem. *Journal of Scheduling*, 9(4):389–396, 2006.

[23] A. S. Schulz. Scheduling to Minimize Total Weighted Completion Time: Performance Guarantees of LP-based Heuristics and Lower Bounds. In *Proc. of the* 5^{th} *Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 301–315, 1995.

[24] A. Snavely and D. M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreading processor. In *Proc. of the 9th Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000.

[25] Standard Performance Evaluation Corporation. *SPEC CPU2006*. `http://www.spec.org/cpu2006/`.

[26] C. S. Sung and S. H. Yoon. Minimizing Total Weighted Completion Time at a Pre-Assembly Stage Composed of Two Feeding Machines. *International Journal of Production Economics*, 54:247–255, 1998.

[27] G. Wang and T. C. E. Cheng. Customer Order Scheduling to Minimize Total Weighted Completion Time. In *Proc. of the* 1^{st} *Multidisciplinary Conference on Scheduling Theory and Applications*, pages 409–416, 2003.

[28] L. A. Wolsey. Mixed Integer Programming Formulations for Production Planning and Scheduling Problems. In *Invited talk at 12th ACM-SIAM Symposium on Mathematical Programming*, 1985.