CBM: Core Based Memory Scheduling method

KOHEI HOSOKAWA1  YASUO ISHII1,2  MARY INABA1  KEI HIRAKI1

Abstract: In modern chip-multiprocessor systems, DRAM is shared among multiple threads. The memory scheduler must resolve the inter-thread contention for the DRAM effectiveness. Previously proposed DRAM memory schedulers have calculated the memory access intensity of each thread for the priority scheduling. Existing methods[2], [3] analyze the number of memory requests served in memory controller to get memory-intensity, but these methods lack prediction accuracy. TCM[1] avoids this problem by using MPKI information. TCM can improve the prediction accuracy, but takes very long cycles to update the thread priority. As a result, TCM lacks the timeliness of the priority prediction. This paper presents a new memory scheduling method, Core-Based Memory scheduling(CBM), which utilizes core information for memory-intensity evaluation of each thread. Our key idea is 1) to refer the distance of instruction count between each memory request for the priority calculation, and 2) to place the priority scheduler on each core. CBM judges the core calculation progress by comparing the instruction counter distance between the new memory request and the last one. By doing so, CBM can utilize the instruction progress information of each core directly, thus we can predict the memory-intensity more accurately.

CBM also proposes to calculate the thread priority not on the memory controller but on the private cache of each core. By doing so, even in concurrent many-channel memory system, CBM can decide priority without the heavy inter-channel communication. Therefore, CBM accomplishes high timeliness on the priority update.

We evaluate CBM by using the workloads of Memory Scheduling Championship (MSC) and compare its performance to two existing scheduling algorithms. We found that CBM achieves both the best throughput and fairness.

Keywords: Memory Scheduling, Prediction

1. Introduction

Off-chip DRAM memory has been one of the major bottlenecks of processing due to the higher latency compared to CPU. This heavy latency gets even longer in chip-multiprocessor (CMP) systems, in which DRAM memory is shared among multiple threads. The inter-thread contention in CMP makes the DRAM performance worse: each thread issues memory requests simultaneously, which wastes spatial and temporal locality with each other and even causes starvation. To enhance the modern DRAM memory system efficiently, it is inevitable to handle this inter-thread contention.

There are three metrics mainly used to evaluate the effectiveness of memory scheduling algorithms: fairness, system throughput, and energy consumption. First, the fairness is important for the multi-thread workloads, otherwise the most delayed thread will slow down the overall system performance. Second, the energy consumption should be low because the DRAM energy consumption is not negligible on the modern CMP systems. Of course, the system throughput should be high for the effective DRAM usage.

To achieve the goal of high system throughput and high fairness, several scheduling algorithms have been proposed. State of the art schedulers exploit the memory-intensity of threads to improve system throughput. In these methods, memory scheduler takes the strategy of prioritizing the memory-non-intensive threads over the memory-intensive threads to reduce the total core stall time in overall system while keeping fairness high. However, when updating the thread priority, existing methods[1], [2], [3] need heavy inter-channel communication periodically to gather the statistical information distributing among every channel. This long time communication and bandwidth overhead restricts the priority update frequency low, resulting in the lack of priority prediction timeliness. Moreover, [2], [3] use the Served Requests Per Cycle (SRPC) on the memory scheduler to estimate the memory-intensity per instruction of each thread. This SRPC-based strategy cannot distinguish the core calculation time and stall time, so the estimated memory-intensity per instruction number is not accurate.

Our new scheduling algorithm exploits the core information of each thread to calculate the thread priority based on two key strategies. First, we use the instruction count distance of each memory request to calculate the memory-access-intensity. We can evaluate accurate memory-intensity by referring the instruction count because the instruction count describes the core progress accurately. Therefore, our scheduler can improve the accuracy of memory-intensity estimation. Second, we move the priority scheduling system from memory scheduler to private cache of each core. By placing priority scheduler on core side, the priority scheduler can detect all the memory requests from a core. This means that the priority scheduler does not need to communicate all memory channels to gather statistical information of

1  The university of Tokyo  
2  NEC

© 2012 Information Processing Society of Japan
memory requests from a core. Without heavy inter-channel synchronization communication, CBM can update thread priority on every memory access happening, enabling more timely priority control.

In this paper, we make the following contributions: 1) We use the "instruction counter distance" of each memory access for the calculation of recent memory-intensity. This method can capture the immediate change of the thread’s memory-intensity state between non-memory-intensive phase and memory-intensive phase. 2) We change the placement of priority scheduler from memory scheduler to private cache of each core. This avoids the heavy inter-channel communication of gathering statistical information for priority scheduling. Therefore, we can update the thread priority at the occurrence of each memory request without suffering inter-channel priority inconsistency.

2. Background and Motivation

2.1 Memory Access statistics

Previous paper has showed that system throughput improves by prioritizing low memory-intensity threads. Low memory-intensity thread spends most of the time in calculation, so the instruction throughput(Instruction Per Cycle) is high. In such case, by prioritizing its memory requests and serving them with low latency, the thread progresses fast. On the other hand, memory-intensive thread progresses relatively smaller per a memory request and stalls immediately. In such case, the memory-intensive thread will progress by rather enhancing row-hit rate and memory access throughput than improving latency low. Therefore, it is important to prioritize non-memory-intensive thread for high system throughput.

Previous schedulers evaluated memory-intensity by analyzing the number of SRPC(served requests per cycle) or MPKI(miss per kilo instruction). The figure 1 shows the MPKI data in Blacksc-holes workload. In figure 1, we can see that the memory-intensity changes between heavy and light for the progress of instruction counter. By prioritizing a thread during the non-memory-intensive phase, the thread will progress far without stalling due to the long LLC miss.

This MPKI tendency is hard to be recognized by analyzing the number of SRPC on memory controller. The figure 2 shows the relationship between SRPC and the number of threads running at a time. SRPC result gets spread uniformly as the thread number increases, and memory-intensity tendency gets hard to be recognized. Moreover, in figure 2(e), there happens a blank-request time caused by the core stall time due to the inter-thread interference. This blank time is to be judged as memory-intensive phase, but the judgment is difficult in the multi-channel memory scheduler (We will show more details in the next subsection). As a result, memory-intensity tendency is unrecognizable by using SRPC information and not using the core stall information or MPKI.

2.2 Inter-channel communication and timeliness

Modern DRAM memory has multiple memory channels separated from each other, so the memory requests which arrive at a memory channel are invisible to the other memory channels. [2] showed that inter-channel synchronization between memory channels is important for the high system throughput. For example, even if a channel A serves thread P eagerly, but if channel B does not prioritize thread P, then the final progression of thread P is delayed by channel B. In such situation, the system throughput will get worse due to the delayed threads besides thread P in channel A. Taken together, prioritizing a thread without synchroniz-
CBM is a fine-grain thread priority scheduling method based on core information. CBM improves the timeliness and accuracy of the priority prediction of Thread Cluster Memory scheduler (TCM, [1]). CBM consists of two separate modules: priority scheduler and memory controller.

CBM places the priority scheduling module on each core (we define the core to which the priority scheduler attached as the "home core"), not on memory scheduler. Priority scheduler has two registers: the last instruction counter (LIC) and the burst counter (BC). LIC stores the instruction counter value (which is the number of the committed instructions of the home core) of the last LLC miss request derived from the home core. When a new memory request occurs, priority scheduler calculates the distance between the current instruction counter and LIC (We call this distance "LIC distance"). If LIC distance exceeds Non-Memory-Intensive Distance Threshold, the priority scheduler considers that the next memory request is issued after a long calculation period from the last LLC miss. Then, the priority scheduler judges that the home core is in the non-memory-intensive phase.

On the other hand, the BC counts the number of sequential memory requests issued in a short while. If LIC distance is smaller than the Memory-Intensive Distance Threshold, BC counts up. If LIC distance exceeds Non-Memory-Intensive Distance Threshold, BC is set 0. When the BC value exceeds Memory-Intensive Burst Threshold, the priority scheduler considers that the home node is in memory-intensive phase.

When a request misses private cache, priority scheduler predicts memory-intensity of the current thread based on LIC and BC value. Then, priority scheduler adds 1-bit Memory-intensity flag to the memory request. This flag is sent to upper level cache and main memory. As a result, even if there are several LLC modules or several memory channels in the system and the requests from a core are distributed among them, there is no need to gather the statistical information from them. Similar to the Memory-intensity flag, 1-bit LLC-Miss flag is attached to the memory access response back to the private cache. When a private cache miss request causes LLC miss and memory access, LLC-Miss flag is set 1. If the request hits any cache, then LLC-Miss flag is set 0. Therefore, priority scheduler on each core can track LLC miss of each request, and also update LIC and BC registers.

As is mentioned above, the LLC miss detection and LIC update happens when a result of LLC miss access returns to home core. Priority scheduler cannot judge whether in-flight memory request causes LLC miss and memory access, LLC-Miss flag is set 1. If the request hits any cache, then LLC-Miss flag is set 0. Therefore, priority scheduler on each core can track LLC miss of each request, and also update LIC and BC registers.
requests are the LLC miss requests or not. For this reason, the memory-intensity prediction gets inaccurate to some extent while waiting for any in-flight request. However, the error range due to this factor is as large as the capacity of the ROB of home core at most. The capacity of modern ROB is generally from 32 to 128, so this error range is not so large for the memory-intensity prediction.

4. Implementation

Figure 5 shows the block diagram of CBM scheduler. CBM requires two hardware support: thread-priority scheduler on each core and memory scheduler on each memory channel as described. The major hardware budget is shown in Table 1. The required hardware storage cost within priority controller is 68bit per core. The additional hardware cost within memory scheduler part is 1bit per read request. The priority scheduler of each core only needs simple calculation, and the calculation can be conducted by snooping the instruction counter before the memory request happens. Therefore, it does not have a large effect on critical path. Each priority scheduler only needs to calculate the requests from home core, so the calculation time does not increase as the core number increases. The memory scheduler part does not need the meta scheduler to gather all information distributing among all channels, so the calculation load of memory scheduler is smaller than the previous scheduler. For this reason, CBM method has scalability to both the channel number and the core number.
We implement Core Based Memory scheduler (CBM) in this paper. CBM separates the priority scheduler module from memory controller, and places it on each core. By this separation, CBM does not need heavy inter-channel communication that was conducted periodically in existing memory scheduling algorithms to gather the memory request statistics. As a result, CBM can update memory-intensity information every memory access, leading to finer-grain thread priority update. Therefore, CBM enhances the timeliness and accuracy of thread priority prediction simultaneously. The priority scheduler on CBM is placed on each core and requires no meta scheduler gathering all statistical information, so it is scalable to the overall core number and the memory channel number in the system. Moreover, memory schedulers in all metrics. CBM reduces the total execution time by 6.2%, the PFP by 14.6%, and the EDP by 12.8% from the baseline FR-FCFS scheduler.

6. Conclusion and Future Works

We proposed Core Based Memory scheduler (CBM) in this paper. CBM separates the priority scheduler module from memory controller, and places it on each core. By this separation, CBM does not need heavy inter-channel communication that was conducted periodically in existing memory scheduling algorithms to gather the memory request statistics. As a result, CBM can update memory-intensity information every memory access, leading to finer-grain thread priority update. Therefore, CBM enhances the timeliness and accuracy of thread priority prediction simultaneously. The priority scheduler on CBM is placed on each core and requires no meta scheduler gathering all statistical information, so it is scalable to the overall core number and the memory channel number in the system. Moreover, memory schedulers in all memory channels can serve memory requests from a core synchronously (or, with the same priority) without inter-channel communication. For this reason, all memory channels can cooperate on CBM scheduler efficiently.

We evaluated CBM scheduler by using the memory scheduling championship framework and its workloads. The experimental

<table>
<thead>
<tr>
<th>Workload</th>
<th>Config</th>
<th>Sum of exec times (10 M cyc)</th>
<th>Max slowdown</th>
<th>EDP (J.s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FR-FCFS</td>
<td>Close</td>
<td>Proposed</td>
<td>FR-FCFS</td>
</tr>
<tr>
<td>MTc</td>
<td>1 chan</td>
<td>398</td>
<td>395</td>
<td>374</td>
</tr>
<tr>
<td>MTc</td>
<td>4 chan</td>
<td>168</td>
<td>158</td>
<td>153</td>
</tr>
<tr>
<td>MTf</td>
<td>1 chan</td>
<td>303</td>
<td>319</td>
<td>305</td>
</tr>
<tr>
<td>MTf</td>
<td>4 chan</td>
<td>238</td>
<td>239</td>
<td>236</td>
</tr>
<tr>
<td>bl-bl-fr-fr</td>
<td>1 chan</td>
<td>147</td>
<td>145</td>
<td>138</td>
</tr>
<tr>
<td>c1-c1</td>
<td>1 chan</td>
<td>82</td>
<td>82</td>
<td>79</td>
</tr>
<tr>
<td>c1-c1</td>
<td>4 chan</td>
<td>52</td>
<td>46</td>
<td>46</td>
</tr>
<tr>
<td>c1-c1-c2</td>
<td>1 chan</td>
<td>230</td>
<td>231</td>
<td>212</td>
</tr>
<tr>
<td>c1-c1-c2</td>
<td>4 chan</td>
<td>124</td>
<td>115</td>
<td>112</td>
</tr>
<tr>
<td>c2</td>
<td>1 chan</td>
<td>43</td>
<td>43</td>
<td>42</td>
</tr>
<tr>
<td>c2</td>
<td>4 chan</td>
<td>30</td>
<td>27</td>
<td>26</td>
</tr>
<tr>
<td>c3-c3-c3-c3-c3-c3-c3-c3</td>
<td>4 chan</td>
<td>210</td>
<td>198</td>
<td>193</td>
</tr>
<tr>
<td>c4-c4-c5-c5</td>
<td>1 chan</td>
<td>126</td>
<td>127</td>
<td>124</td>
</tr>
<tr>
<td>c4-c4-c5-c5</td>
<td>4 chan</td>
<td>71</td>
<td>67</td>
<td>67</td>
</tr>
<tr>
<td>fa-fa-fe-fe</td>
<td>1 chan</td>
<td>215</td>
<td>216</td>
<td>200</td>
</tr>
<tr>
<td>fa-fa-fe-fe</td>
<td>4 chan</td>
<td>102</td>
<td>96</td>
<td>92</td>
</tr>
<tr>
<td>fl-fl-sw-c2-c2-fe-fe</td>
<td>4 chan</td>
<td>282</td>
<td>268</td>
<td>257</td>
</tr>
<tr>
<td>fl-fl-sw-c2-c2-fe-fe</td>
<td>4 chan</td>
<td>618</td>
<td>601</td>
<td>590</td>
</tr>
<tr>
<td>-bl-bl-fr-l1-c1-st-st</td>
<td>4 chan</td>
<td>238</td>
<td>235</td>
<td>220</td>
</tr>
<tr>
<td>-fl-sw-c2-c2</td>
<td>4 chan</td>
<td>127</td>
<td>119</td>
<td>116</td>
</tr>
<tr>
<td>le-le-le-le</td>
<td>1 chan</td>
<td>225</td>
<td>226</td>
<td>210</td>
</tr>
<tr>
<td>le-le-le-le</td>
<td>4 chan</td>
<td>167</td>
<td>149</td>
<td>150</td>
</tr>
<tr>
<td>li-li</td>
<td>4 chan</td>
<td>109</td>
<td>123</td>
<td>119</td>
</tr>
<tr>
<td>li-li</td>
<td>4 chan</td>
<td>94</td>
<td>74</td>
<td>73</td>
</tr>
<tr>
<td>li-li-mu-mu-mu-ti-ti</td>
<td>4 chan</td>
<td>590</td>
<td>548</td>
<td>526</td>
</tr>
<tr>
<td>li-li-mu-mu</td>
<td>1 chan</td>
<td>378</td>
<td>393</td>
<td>398</td>
</tr>
<tr>
<td>li-li-mu-mu</td>
<td>4 chan</td>
<td>215</td>
<td>190</td>
<td>185</td>
</tr>
<tr>
<td>st-st-st-st</td>
<td>1 chan</td>
<td>159</td>
<td>156</td>
<td>150</td>
</tr>
<tr>
<td>st-st-st-st</td>
<td>4 chan</td>
<td>84</td>
<td>80</td>
<td>78</td>
</tr>
<tr>
<td>ti-ti</td>
<td>1 chan</td>
<td>169</td>
<td>162</td>
<td>157</td>
</tr>
<tr>
<td>ti-ti</td>
<td>4 chan</td>
<td>102</td>
<td>89</td>
<td>87</td>
</tr>
<tr>
<td>Overall</td>
<td>6188</td>
<td>6006</td>
<td>5804</td>
<td>5.14</td>
</tr>
</tbody>
</table>

Table 2 Comparison of key metrics on baseline and proposed memory controllers.
result showed that CBM scheduler improves the total execution time by 6.2%, the PFP by 14.6%, and the EDP by 12.8% from the baseline FR-FCFS scheduler respectively.

CBM method can be also applied to the case that the transfer of memory requests will be irregularly delayed, such as Network on Chip (NoC) structure. The request priority is set by the priority scheduler on each core, so all requests have already known their own priority when they are issued from home core. Therefore, these priority information can be used for the routing algorithm of NoC for example. Combining CBM scheduling method with the NoC routing algorithm is our future works.

References