GDP: Using Dataflow Properties to Accurately Estimate Interference-Free Performance at Runtime

Magnus Jahre
Department of Computer Science
Norwegian University of Science and Technology (NTNU)
Email: magnus.jahre@ntnu.no

Lieven Eeckhout
Department of Electronics and Information Systems
Ghent University
Email: lieven.eeckhout@ugent.be

***Abstract*—Multi-core memory systems commonly share resources between processors. Resource sharing improves utilization at the cost of increased inter-application interference which may lead to priority inversion, missed deadlines and unpredictable interactive performance. A key component to effectively manage multi-core resources is performance accounting which aims to accurately estimate interference-free application performance. Previously proposed accounting systems are either *invasive* or *transparent*. Invasive accounting systems can be accurate, but slow down latency-sensitive processes. Transparent accounting systems do not affect performance, but tend to provide less accurate performance estimates.**

We propose a novel class of performance accounting systems that achieve both performance-transparency and superior accuracy. We call the approach *dataflow* accounting, and the key idea is to track dynamic dataflow properties and use these to estimate interference-free performance. Our main contribution is *Graph-based Dynamic Performance (GDP)* accounting. GDP dynamically builds a dataflow graph of load requests and periods where the processor commits instructions. This graph concisely represents the relationship between memory loads and forward progress in program execution. More specifically, GDP estimates interference-free stall cycles by multiplying the critical path length of the dataflow graph with the estimated interference-free memory latency. GDP is very accurate with mean IPC estimation errors of 3.4% and 9.8% for our 4- and 8-core processors, respectively. When GDP is used in a cache partitioning policy, we observe average system throughput improvements of 11.9% and 20.8% compared to partitioning using the state-of-the-art Application Slowdown Model.

***Keywords*-Multi-core memory systems; Accounting**

I. Introduction

Chip Multi-Processors (CMPs) commonly share memory system resources. This is often beneficial since it leads to improved utilization, but it also makes destructive interference possible [1]. Consequently, the performance of an application may be influenced significantly by its co-runners. This lack of *performance predictability* may be an annoyance to the desktop user, but it can be a business-critical issue for data center operators [2]. Furthermore, OS scheduling policies commonly assume that the progress made by each process is independent of the behavior of other processes. Destructive interference breaks this assumption and may cause problems such as missed deadlines, priority inversion, unpredictable interactive performance, and not complying with service-level agreements [3].

Destructive interference can be reduced by techniques implemented in both software and hardware. These techniques require accurate estimates of how destructive interference affects the applications that are currently running. A performance accounting system provides architectural support for estimating interference-free or private mode[2] performance. This is a challenging task because modern processors contain a variety of latency-hiding mechanisms (out-of-order execution, non-blocking caches, etc.). Performance accounting systems have wide applicability, and have previously been used in interference-aware OS schedulers (e.g., [3, 4, 5, 6]) and in a variety of shared memory system resource management systems (e.g., [7, 8, 9, 10, 11, 12, 13]).

Previously proposed accounting schemes can be broadly partitioned into two classes: *invasive* and *transparent*. Invasive accounting systems [14, 15, 16] estimate private mode performance by changing architectural policies such that interference is minimized for a single process over a short period of time. For example, the *Application Slowdown Model (ASM)* [14] periodically gives a single process high priority in the memory controller and uses simple policies to account for the remaining sources of interference. This approach can provide accurate performance estimates, but may slow down latency-sensitive processes. In our evaluation, we have observed up to 57% performance reduction for individual processes compared to a configuration without accounting.

Transparent accounting systems [3, 4, 17] do not affect application performance. Prior techniques are *architecture-centric* and use a set of pre-defined conditions regarding architectural state to determine if a cycle of memory latency is due to interference and whether it will increase

[2]The *private mode* is a configuration used for off-line evaluation where the application runs on one core of the CMP and all other cores are idle. Thus, the application has exclusive access to all shared resources. In *shared mode*, all cores are active and the applications compete for shared resources.

processor stall cycles. Examples of monitored conditions are a full *Re-Order Buffer (ROB)*, stalled register rename stage or if current loads access the shared memory system. Architecture-centric systems are typically less accurate than invasive systems. First, they are unable to accurately account for *Memory-Level Parallelism (MLP)* [14]. This can result in a single interference event being accounted multiple times. In addition, they can only account for interference effects that match the pre-defined conditions. Selecting conditions are difficult because it highly depends on the application which CPU structures become performance bottlenecks.

In this work, we propose a novel transparent accounting system that provides significantly higher accuracy by taking a radically different approach which we call *dataflow* accounting. In this approach, we exploit that the application and the underlying architecture are identical in the shared and private modes. Therefore, resource bottlenecks are the same in the two modes, but the performance impact differs due to longer shared mode memory latencies.

Our *Graph-based Dynamic Performance (GDP)* accounting technique exploits this insight. GDP detects dependencies between loads and periods where the processor commits instructions and uses this to dynamically create a dataflow graph. This graph models the relationship between load requests and forward progress during program execution. Since loads are serviced in parallel, the *Critical Path Length (CPL)* of the dataflow graph determines the number of non-overlapped loads needed to execute the program. Thus, the number of load-related stall cycles can be estimated by multiplying CPL with the estimated average private mode memory latency. We also propose a variant of GDP called *Graph-based Dynamic Performance with Overlap (GDP-O)*. GDP-O extends upon GDP by accounting for the overlap between processor commit periods and memory loads.

To illustrate that high accuracy leads to improved system performance, we develop a *Last-Level Cache (LLC)* management policy which we call *Model-based Cache Partitioning (MCP)*. Like prior work (e.g., [8, 14]), MCP uses *Auxiliary Tag Directories (ATDs)* to obtain private mode miss curves for each process and enforces allocations with way-partitioning. MCP uses a first-order performance model to estimate shared mode performance as a function of LLC misses. The private mode performance estimates provided by GDP or GDP-O enable MCP to directly optimize for system performance metrics such as *System Throughput (STP)* [18]. This is especially useful when the LLC management policy needs to choose between the working sets of different processes. In this case, accurate private mode performance estimates empower MCP to select the working sets that will maximize the chosen system-level performance metric.

In summary, the contributions of this work are:

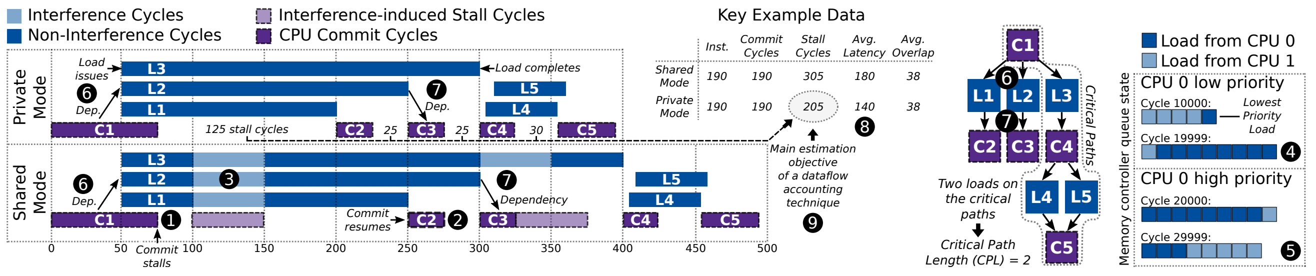
- We propose *dataflow performance accounting* which uses dynamic shared mode dataflow properties to provide significantly more accurate private mode performance estimates than prior work.
- We propose the GDP dataflow accounting technique which is based on the observation that the number of loads on the critical path through the load and commit period dependency graph determines the number of non-overlapped memory loads necessary to execute the program. GDP is very accurate and reduces private mode performance *Root Mean Squared (RMS)* error by 7.4× and 7.7e12× compared to invasive ASM accounting [14] for our 4- and 8-core CMPs, respectively.
- We propose the GDP-O accounting technique that extends upon GDP by accounting for the overlap cycles where the processor commits instructions while at least one load is pending. GDP-O reduces the stall cycle RMS error by 13.5% and 10.8% compared to GDP for our 4- and 8-core CMPs, respectively.
- We propose the LLC management policy MCP to establish that accurate private mode performance estimates can be leveraged by resource management policies to improve system performance. MCP improves average system throughput by 11.9% and 20.8% compared to state-of-the-art ASM cache partitioning [14] for our 4- and 8-core CMPs, respectively.

II. Explaining Dataflow Accounting's Accuracy

Performance accounting benefits greatly from a model that leverages the similarities between the shared and private modes. Although the CPU executes the same application in both modes, it is challenging to reliably detect these similarities because memory request timing can change significantly. Figure 1a shows the same sequence of memory loads and periods where the processor commits instructions in the two modes. The out-of-order CPU executes instructions when their dependencies are satisfied. It commits an instruction when its computation is complete and prior instructions have committed. For load instructions, execution entails issuing a memory request. If a load instruction reaches the head of the ROB before the memory request completes (e.g., *L1* at ❶), the CPU stops committing instructions or *stalls*. When the memory request completes, the CPU commits one or more instructions. For example, memory load *L1* enables the instructions in commit period *C2* to commit before *L2* reaches the head of the ROB and causes another stall at ❷.

Figure 1a shows that loads *L1*, *L2* and *L3* are delayed by interference which causes longer memory latencies in the shared mode (e.g., at ❸). These loads are serviced in parallel, and the amount of parallelism changes throughout the load burst. Accurate MLP modeling is a critical component of a performance accounting system since MLP determines the performance impact of increased memory latencies.

Architecture-centric accounting: The best architecture-centric accounting scheme is PTCA [3] which assumes that the private mode CPU stalls are the interference cycles



Key Example Data

	Inst.	Commit Cycles	Stall Cycles	Avg. Latency	Avg. Overlap
Shared Mode	190	190	305	180	38
Private Mode	190	190	205	140	38

(a) Dataflow accounting (b) Dataflow graph (c) Invasive accounting

Figure 1: Example explaining why dataflow accounting is significantly more accurate than prior work

a load request is subjected to while the ROB is full. In Figure 1a, the CPU is able to keep the ROB practically full. Thus, PTCA assumes that all interference cycles reduce stall length.

PTCA processes loads independently, and this can cause inaccuracies when a single interference event affects multiple loads as shown at ❸. In this case, PTCA will correctly reduce the length of the stall between *C1* and *C2*, but incorrectly estimate that the stall between *C2* and *C3* did not occur in the private mode. The reason is that the interference latency of *L2* is greater than the shared mode stall. However, the stall did occur in the private mode because it is caused by serialization in the memory controller. Our experiments, which are discussed in detail in Section VII, shows that PTCA is inaccurate with relative RMS performance estimation errors of 103.3%, 78.8% and 81.5% for our 2-, 4- and 8-core CMPs, respectively.

Invasive accounting: Invasive accounting schemes such as ASM [14] avoid modeling MLP. Instead, ASM gives each CPU highest priority in the memory controller for a few thousand clock cycles (i.e., an *epoch*) to minimize interference. The assumption is that the performance of the application in the high priority epoch will be similar to its private mode performance.

Unfortunately, limiting interference does not recreate the situation that occurs in the private mode. Figure 1c illustrates this with a 2-core CMP running two memory-intensive applications over two epochs. At the end of the first epoch, CPU 0 has accumulated a significant backlog of memory requests due to running with low priority at ❹. CPU 0 then spends the subsequent epoch trying to clear this backlog which causes a backlog for CPU 1 at ❺. These backlogs do not occur in the private mode and leads ASM to significantly overestimate private mode memory controller queuing delays. The problem is exacerbated by adding more cores to the system since this increases the time between high priority epochs. Our experiments show that ASM can be inaccurate with relative RMS performance estimation errors of 33.7%, 187.2% and 1.34e14% for our 2-, 4- and 8-core CMPs, respectively.

Dataflow accounting: Prior transparent accounting techniques are inaccurate because they are unable to accurately model private mode MLP. To overcome this problem, we propose dataflow accounting. The first key observation is that the dataflow dependencies between memory loads and commit periods are very similar in the shared and private mode. The reason is that all load requests occur as a response to an instruction being committed. Concretely, the load depends on the memory-related instruction that enabled the execution of the instruction dependency chain that eventually allowed the load to execute. Conversely, all completed loads lead to at least one instruction committing. The minimal case is that only the load instruction itself can commit.

The dataflow graph can be constructed dynamically based on two simple rules: (1) The parent of a load request is the commit period that started closest in time before the request was issued; and (2) The child of a memory request is the commit period that finished closest in time after the request completed. Figure 1b shows the dependency graph resulting from applying these rules to the shared mode loads and commit periods. For example, *C1* unlocks the instructions that allow *L2* to meet its dependencies and issue its memory request at ❻. When *L2* completes, it allows the instructions in *C3* to commit at ❼. The (*C1*, *L2*) and (*L2*, *C3*)-dependencies are the same in the two modes.

The second key observation is that the dataflow graph represents the relationship between memory loads and periods where the CPU commits instructions. Thus, the number of memory loads on a critical path of the dependency graph indicates the number of non-overlapped loads necessary to execute the program. For example, the dataflow graph in Figure 1a shows that the critical loads are *L3*, *L4* and *L5*, and that *L4* is executed in parallel with *L5*. The private mode stall cycles can then be estimated by multiplying the number of loads on a critical path by the estimated private mode memory latency. We design a novel, low-overhead hardware unit that estimates the *Critical Path Length (CPL)* using an approximation of Kahn's algorithm [19].

In the next two sections, we will explain in detail how our GDP and GDP-O schemes compute private mode performance estimates. First, we develop an analytical performance model that formalizes the performance impact of memory system interference in Section III. This model forms the foundation for GDP and GDP-O. Then, we explain GDP and GDP-O in detail in Section IV.

III. Modeling Interference

The accounting systems we propose in this paper are based on the analytical performance model by Karkhanis and Smith [20] and the observation that performance can be modeled by quantifying steady-state performance (i.e., perfect branch predictor and perfect memory system) and then subtracting the performance loss due to each imperfect component.

The performance P_p of process p is the clock cycles where the process committed one or more instructions (C_p) plus the cycles where commit was stalled (S_p) divided by the number of committed instructions (Inst_p):

$$\mathrm{CPI}_p = P_p = (C_p + S_p^{\mathrm{Ind}} + S_p^{\mathrm{Loads}} + S_p^{\mathrm{Other}})/\mathrm{Inst}_p \quad (1)$$

The total stall cycles is the sum of the memory independent stall cycles (S_p^{Ind}), stall cycles due to load instructions (S_p^{Loads}) and other memory-related stalls (S_p^{Other}). From this definition, a processor is stalled when it is not committing instructions. However, it can do useful work in the background. The key insight is that this work cannot result in forward progress until the instruction that blocks commit is completed.

Since the only difference between the shared and private mode is the memory system behavior, S_p^{Ind} is by definition the same in the two modes. Through systematic empirical analysis we determined that the memory-dependent stall cycles are due to four events. The most important factor is the stall cycles that occur when a load instruction reaches the head of the ROB (S_p^{Loads}). Such stalls have two distinct causes: load requests that remain in the private memory system of the CMP (S_p^{PMS}) and loads that visit the shared memory system of the CMP (S_p^{SMS}). We refer to these load types as *Private Memory System Loads (PMS-loads)* and *Shared Memory System Loads (SMS-loads)*. PMS-loads cannot be directly influenced by interference since they do not access shared units. Thus, memory system interference primarily affects performance through SMS-loads.

S^{Other} is the sum of three rare events. The first stall type occurs when the store buffer is full at the same time as a store instruction is at the head of the ROB. The second type is due to the L1 data cache being blocked because of too many in-flight memory requests. This causes a stall when a load that needs to access the blocked L1 cache reaches the head of the ROB. Finally, commit might stall when a branch mispredict resolves and the ROB only contains wrong-path instructions. Assuming that the stall length is proportional to the memory latency difference between the shared and private modes is sufficiently accurate for these events.

Equation 2 ties these observations together and describes how we compute our private mode performance estimates $\hat{\pi}_p$ from the stall estimates $\hat{\sigma}_p^{\mathrm{SMS}}$ and $\hat{\sigma}_p^{\mathrm{Other}}$:

$$\hat{\pi}_p = (C_p + S_p^{\mathrm{Ind}} + S_p^{\mathrm{PMS}} + \hat{\sigma}_p^{\mathrm{SMS}} + \hat{\sigma}_p^{\mathrm{Other}})/\mathrm{Inst}_p \quad (2)$$

We use Greek letters for private mode values and a hat to show that a value is a shared mode estimate of a private mode value (e.g., $\hat{\sigma}$). In other words, σ_p^{SMS} is the private mode counterpart of the shared mode SMS-stall cycles S_p^{SMS}.

IV. Estimating Private Mode Stalls

In this section, we describe our dataflow accounting system in detail. We first present how GDP and GDP-O can be implemented in Section IV-A. Then, we discuss private mode memory latency estimation in Section IV-B and performance and area overheads in Section IV-C.

A. Implementing GDP and GDP-O

We use Figure 1a to show how GDP and GDP-O use the dependency graph to estimate private mode performance. Both GDP and GDP-O base their estimates on the performance model in Equation 2. The example in Figure 1a contains no memory independent stall cycles, no PMS-loads and no other stalls (i.e., S_p^{Ind}, S_p^{PMS} and $\hat{\sigma}_p^{\mathrm{Other}}$ are equal to 0). Furthermore, there are 190 committed instructions (Inst_p) and 190 commit cycles (C_p). The purpose of GDP and GDP-O is to estimate the SMS-load-related stall cycles ($\hat{\sigma}_p^{\mathrm{SMS}}$). For simplicity, we assume a perfect private mode memory latency estimator (Section IV-B removes this restriction).

GDP estimates SMS-load-related stall cycles by multiplying CPL with the estimated average private mode memory latency (i.e. $\hat{\sigma}_p^{\mathrm{SMS-GDP}} = \mathrm{CPL}_p \cdot \hat{\lambda}_p$). There are two SMS-loads on the critical paths of Figure 1a's dataflow graph which gives a CPL of 2. The private mode latency estimator provides GDP with a perfect estimate of 140 cycles ❽. Thus, GDP estimates that the number of SMS-load-related stall cycles is 280 (i.e., $\hat{\sigma}_p^{\mathrm{SMS-GDP}} = 2 \cdot 140$). This estimate is higher than the actual number of private mode stall cycles which is 205 ❾. By inserting 190 instructions, 190 commit cycles and 280 estimated SMS-load stall cycles into Equation 2, GDP estimates that the private mode CPI is 2.5 ($[190 + 280]/190$) while the actual value is 2.1.

GDP's private mode performance estimate is not perfect because it does not account for the overlap between pending loads and commit periods. For example, *L1*, *L2* and *L3* overlap with *C1* at ❶. GDP-O removes this error by subtracting the estimated number of cycles where the CPU is committing instructions while at least one SMS-load is pending (i.e., $\hat{\sigma}_p^{\mathrm{SMS-GDP-O}} = \mathrm{CPL}_p \cdot [\hat{\lambda}_p - O_p]$). However, we do not know if a request is an SMS-load or a PMS-load until it completes. Thus, we count the number of overlapped cycles for all L1 load misses. When an SMS-load completes, we add the per-request counter to a global

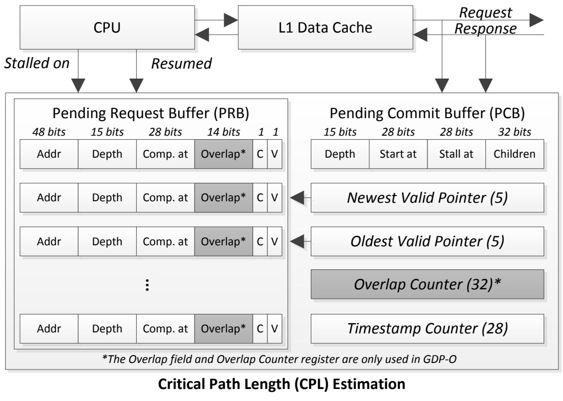


Figure 2: Dataflow graph processing architecture

Algorithm 1: Load Request Issued

```
if PRB[newestValidPointer] is valid then
    Increment newestValidPointer (wrap around if necessary);
    if newestValidPointer == oldestValidPointer then
        Invalidate PRB[newestValidPointer];
Add request to the PRB and as child of pending commit;
```

Algorithm 2: Load Request Completed

```
Find request in PRB;
if Request not found then
    return;
if Request has been in the shared memory system then
    Set completed bit and update the Completed at field;
else
    Invalidate PRB entry and remove PCB pointer;
```

overlap counter. When the estimate is computed, we divide the global overlap counter by the number of SMS-loads to compute the average overlap O_p. Since the overlap is a property of the program (e.g., number of instructions between SMS-loads) and processor core architecture (e.g., entries in the ROB), O_p is similar in the two modes.

In Figure 1a, the average overlap is 38 cycles. Thus, GDP-O estimates that the number of SMS-load-related stall cycles is 204 (i.e., $\hat{\sigma}_p^{\mathrm{SMS-GDP}} = 2 \cdot [140 - 38]$). Using Equation 2, GDP-O estimates that the private mode CPI is 2.1 ($[190 + 204]/190$) which is equal to the actual private mode CPI.

Dataflow Graph Processing: We use two hardware structures called the *Pending Commit Buffer (PCB)* and the *Pending Request Buffer (PRB)* to monitor the dependencies between commit periods and SMS-load requests. The PCB is a register and the PRB is a fully associative buffer indexed by the request address. In addition, the PRB can be accessed by the buffer index. Figure 2 shows that these buffers are placed close to the processor core and the L1 data cache. All dependency graph processing is off the critical path of the processor and has no performance impact.

Figure 2 shows the fields of the PCB and the PRB. The PCB field *Depth* stores the current CPL value, and the *Started at* and *Stalled at* fields are the timestamps when the current commit period started and completed, respectively. The timestamps are generated by a cycle counter which is reset when the CPL is retrieved. Finally, the *Children* field is a bit vector of the same length as there are entries in the PRB. If a bit in this vector is set, it means that the pending load at that index in the PRB is a child of this commit period. The *Address* field is the physical address of the memory request, and the *Depth* field is the CPL value for this request. The *Completed* flag is used to tag a request as completed and shared. The *Valid* bit is set when the buffer entry is in use. Finally, *Completed at* is the timestamp when the request completed and *Overlap* is the number of clock cycles the processor committed instructions while this request was pending. Figure 2 also shows the number of bits required for each register field. The bit widths of these fields were selected to accommodate the largest values observed in our experiments. With 32 PRB entries, the storage overhead of GDP and GDP-O are 3117 and 3597 bits, respectively.

Dependency Graph Construction: Algorithm 1 is executed when a load request misses in the L1 cache. If the buffer contains at least one request, we increment the pointer and check if the buffer is full. If it is, we invalidate the oldest pending request. We show that this simple policy does not significantly affect private mode CPL estimation accuracy in Section VII-B. Finally, the request is added to the PRB and to the child list of the current commit in the PCB.

Algorithm 2 shows the operations carried out when an L1 miss completes. We first check if the request is still in the PRB by using its address. If it is not found, it has been evicted due to lack of space. If the request has been in the shared memory system, we set the completed bit and update the timestamp. A request is known to be an SMS-load when the response arrives at the last-level private cache. This information is propagated with the response to the CPL estimation unit. If the request is a PMS-load, we invalidate it and remove the PCB pointer if necessary. Dependencies between PMS-loads and SMS-loads are handled through intervening commit periods.

Algorithm 3 is run when the processor resumes execution after a stall. First, we use the address of the load instruction that caused the stall to access the PRB. If this is a miss, the request was either a PMS-load or removed from the buffer because of limited buffer space. In both cases, we assume this was a PMS-stall which does not affect the CPL. If the address is found, the stall was due to an SMS-load. In this case, we carry out two steps. First, we process and invalidate all requests in the shared buffer that completed before the stall. These requests are the parents of the previous commit period. The depth of the completed commit period is the maximum depth of all its predecessors. Then, we set the *Depth* of all children of the current commit to the depth of the commit plus one. Step 2 initializes the new commit

Algorithm 3: CPU Resumed

```
Find the request $s$ that caused the stall in the PRB;
if $s$ is found then
    // Step 1: Complete commit period $l$
    for Completed request $r$ in the PRB do
        if $r$ completed before the stall then
            if $r.depth > l.depth$ then
                $l$.depth = $r$.depth;
            Invalidate completed request $r$;
    for All children $c$ of the last commit period $l$ do
        $c$.depth = $l$.depth + 1;
    // Step 2: Initialize commit period $p$
    Initialize the new commit period $p$ with depth $s$.depth;
    for Completed request $r$ in the PRB do
        if $r.depth > p.depth$ then
            $p$.depth = $r$.depth;
        Invalidate completed request $r$;
```

period. First, we initialize the PCB by setting the *Started at* timestamp to the current value and resetting the *Stalled at* field and the *Children* bits. Then, we initialize the PCB *Depth* to the depth of the request that caused the stall. Finally, we check if other completed requests have a larger depth value and then invalidate them. The *Stalled at* field of the PCB is set when the processor stalls on a load instruction.

Computing the Dataflow Graph CPL: Our algorithms ensure that the PCB always contains the CPL at the time when the current commit period started. The reason is that the algorithms collectively implement Kahn's algorithm for computing the length of a critical path of a directed acyclic graph (DAG) [19]. Kahn's approach is to first topologically sort the nodes and then traverse the DAG in topological order. In this case, the CPL of all predecessors is computed by the time a node is processed, and the CPL of that node is the maximum of its predecessors plus one. The dependency graph is a DAG when the depth is incremented in Algorithm 3 because the first loop removes all cycles.

We use time to topologically sort the load requests. By the time the depth of a commit period is updated, all requests that can influence its depth are complete. Conversely, the commit period that can influence the depth of a request is complete by the time the request is completed. These properties are true by construction. For instance, if the request is not complete when commit period c_1 ends, it is in fact the parent of the next commit period c_2 and thus cannot influence the critical path length of commit period c_1.

Buffering all SMS-loads that may influence the CPL will in certain cases be infeasible. An example of such a situation is a sequence of shared cache hits separated by a sufficient number of memory-independent instructions. In this case, the number of nodes in the graph will increase but the critical path length will remain the same. This observation is the main motivation for our simple PRB management scheme. If the oldest issued load has not caused a stall, it is unlikely that it will increase the CPL.

B. Estimating Private Mode Latencies

We use the *Dynamic Interference Estimation Framework (DIEF)* [21] to provide private mode latency estimates. DIEF measures the shared mode memory latency L_p and estimates the latency due to inter-process interference I_p. Then, it estimates the private mode latency λ_p by subtracting the interference estimate from the shared mode latency:

$$\lambda_p = L_p - I_p \tag{3}$$

To estimate interference, DIEF uses strategically positioned counters in the on-chip interconnect, LLC and memory controller. In the interconnect, DIEF counts the additional latency a request experiences when delayed by a request belonging to a different process. In the LLC, DIEF uses ATDs [8] to identify interference-induced misses and then records the miss penalty of these requests. DIEF's memory bus interference estimator supports out-of-order request scheduling and accounts for the large latency variation caused by the row-buffers and banks in modern DRAMs. DIEF uses a low-overhead hardware mechanism to emulate the out-of-order scheduling algorithm and thereby estimates the private mode service order and latency of each request.

The original DIEF proposal used full-map tag directories which has a storage cost that we would like to avoid. To achieve this, we employ set sampling [22]. Here, only a small number of sets are stored in the ATD and it is assumed that these are representative for the other sets. Adding support for set sampling reduces the storage overhead of DIEF from 929 KB, 1859 KB and 7178 KB to 5.0 KB, 9.9 KB and 23.8 KB for our 2-, 4- and 8-core CMPs, respectively.

C. Performance and Area Overhead

GDP and GDP-O do not have a direct performance overhead since they are implemented in hardware and are off the critical path. However, they could have an indirect performance impact if a delay in providing private mode performance estimates results in untimely quota allocation. With a hardware management policy, the quotas are reevaluated when the shared resource is repartitioned. These events are rare in computer architecture terms with repartitioning being carried out once every one to five million cycles [8, 9, 23]. Computing Equation 2 for GDP requires 2 divisions, 2 multiplies and 5 additions for each processor. If we assume a sequential implementation where an addition takes 1 cycle, a multiply takes 3 cycles and a division takes 25 cycles, it takes 71 cycles to compute one performance estimate. This latency is similar to prior work [3, 4, 14].

The main source of area overhead for GDP and GDP-O is the DIEF private mode latency estimation mechanism. DIEF's main area-driver is the sampled ATDs required to identify interference-induced LLC misses. This overhead is similar to prior work because ASM [14], ITCA [4] and PTCA [3] all use sampled ATDs for this purpose. In addition, GDP and GDP-O require a CPL estimator per core and a number of counters strategically positioned throughout the CMP. The area overhead of these components is less than 2 KB for our 4-core CMP and therefore small compared to the overhead of DIEF (9.9 KB).

V. Model-based Cache Partitioning

To illustrate the benefits of using private mode performance estimates for memory system resource management, we develop an LLC partitioning technique which we call *Model-based Cache Partitioning (MCP)*. Like previous work [8, 14], MCP uses ATDs to obtain private mode miss curves, way-partitioning to enforce cache allocations and select per-application cache quotas at regular intervals (e.g., 5 million clock cycles). Unlike prior work, MCP selects allocations based on dynamic estimates of the system-level performance metric *System Throughput (STP)* [18]. To accomplish this, MCP needs to be able to estimate the per-application performance impact of LLC capacity allocations. MCP allocates LLC capacity at way-granularity, and the ATDs are used to estimate the number of misses an application would experience with a certain number of ways.

Equation 4 shows the performance model we use to link LLC misses to performance:

$$\mathrm{CPI}_p(m_p) = P_p(m_p) = P_p^{\mathrm{PreLLC}} + P_p^{\mathrm{PostLLC}}(m_p) \quad (4)$$

We exploit that the application CPI can be separated into components for different performance loss events [20]. First, P_p^{PreLLC} captures the performance on the CPU-side of the LLC which is the CPI with infinite LLC capacity. Second, P_p^{PostLLC} captures the CPI impact of the LLC misses caused by SMS-loads. We use a linear first-order model to estimate the CPI increase for every additional miss (i.e., $P_p^{\mathrm{PostLLC}} = g_p \cdot m_p$). This is an approximation since it assumes that the amount of MLP remains constant when the number of misses increases. Further, it assumes that the average per-load memory bus queuing latency remains constant when the total number of memory requests changes. Subramanian et al. [15] showed that these assumptions are reasonable for coarse-grain management decisions by demonstrating that performance decreases linearly with increased memory bus interference.

Equation 5 shows how we compute the application CPI component with an infinite LLC:

$$P_p^{\mathrm{PreLLC}} = (C_p + S_p^{\neg\mathrm{SMS}} + \hat{\mathrm{CPL}}_p \cdot \bar{L}_p^{\mathrm{PreLLC}})/\mathrm{Inst}_p \quad (5)$$

It contains the compute and stall cycles in Equation 1 that are incurred on the CPU-side of the LLC[3]. The CPU stall cycles due to SMS-load latencies in the on-chip interconnect and the LLC are estimated by using the measured average pre-LLC latency ($\bar{L}_p^{\mathrm{PreLLC}}$) and a locally computed estimate of the critical path length. We approximate CPL by dividing the stall time by the latency (i.e., $\hat{\mathrm{CPL}}_p = S_p^{\mathrm{SMS}}/L_p^{\mathrm{SMS}}$)[4].

Table I: CMP Model Parameters

Parameter	Value [Multiple value encoding: 2-core/4-core/8-core]
Clock frequency	4 GHz
Processor Cores	128 entry reorder buffer, 32 entry load/store queue, 64 entry instruction queue, 4 instructions/cycle, 4 integer ALUs, 2 integer multiply/divide, 4 FP ALUs, 2 FP multiply/divide, hybrid branch predictor with 2048 local history registers, 4-way and a 2048 entry BTB
L1 Data Cache	2-way, 64KB, 3/3/2 cycles latency, 16 MSHRs
L1 Inst. Cache	2-way, 64KB, 3/3/2 cycles latency, 16 MSHRs
L2 Private Cache	4-way, 1 MB, 9/9/6 cycles latency, 16 MSHRs
L3 Shared Cache	16-way, 8/8/16 MB, 16/16/12 cycles latency, 32/64/128 MSHRs per bank, 4 banks
Ring Interconnect	4 cycles per hop transfer latency, 32 entry request queue, 1/1/2 request rings, 1 response ring
Main memory	DDR2-800, 4-4-4-12 timing, 64 entry read queue, 64 entry write queue, 1 KB pages, 8 banks, FR-FCFS scheduling [24], Open page policy, 1 channel

Equation 6 shows how we compute the CPI gradient g_p:

$$g_p = (\hat{\mathrm{CPL}}_p \cdot \bar{L}^{\mathrm{PostLLC}})/\mathrm{Inst}_p \quad (6)$$

Similarly to the CPU-side SMS-latency calculation, it consists of the locally computed CPL estimate and the average memory controller and bus latency. $\bar{L}^{\mathrm{PostLLC}}$ is the same for all processes since off-chip bandwidth is shared between cores. Intuitively, Equation 6 expresses the increase in CPI for each additional LLC miss that is due to an SMS-load. To simplify the implementation, we measure P_p^{InfLLC} and the current shared mode CPI $P(m_p)$ and use these values to compute P^{PostLLC} with Equation 4.

Equation 7 shows how we use the performance model to create an online estimate of system throughput ($\hat{\mathrm{STP}}$) from SMS-load LLC misses (m_i) in a system with n CPUs:

$$\hat{\mathrm{STP}}(m_0, m_1, \ldots, m_n) = \sum_{i=0}^{n} \hat{\pi}_i/(P_i^{\mathrm{PreLLC}} + g_i \cdot m_i) \quad (7)$$

The estimated LLC misses for a given allocation are retrieved from the ATDs, and the private mode CPI estimates ($\hat{\pi}_i$) are provided by GDP or GDP-O. Thus, Equation 7 provides a layer on top of a miss-minimizing partitioning algorithm that enables making allocation decisions based on system performance rather than misses. MCP uses the lookahead algorithm [8] to select cache quotas.

VI. Methodology

We use an in-house simulator derived from M5 [25] for our experiments. Our CMP has two private cache levels (L1 and L2) and a shared LLC (L3). The per-core memory systems are connected to the shared cache with a ring interconnect. This high-level architecture is inspired

[3] $S_p^{\neg\mathrm{SMS}}$ is the aggregate of all stall cycles that are not related to the shared memory system (i.e., $S_p^{\neg\mathrm{SMS}} = S_p^{\mathrm{Ind}} + S_p^{\mathrm{Other}} + S_p^{\mathrm{PMS}}$).

[4] The CPL estimate also captures the overlap between processor commit cycles and memory request cycles that GDP-O accounts for. However, we do not explicitly show this since it would unnecessarily complicate the mathematical formulation.

by commercial CMP implementations [26]. We faithfully model DDR2 and DDR4 memory bus interfaces [27, 28]. The default parameters of our CMP models can be found in Table I. We analyze the impact of changing LLC and memory controller parameters in Section VII-D.

We use 52 SPEC2000 and SPEC2006 benchmarks to generate multi-programmed workloads [29, 30]. To avoid initialization effects, we fast-forward each benchmark in a single core configuration for 20 billion instructions and take a checkpoint. This checkpoint contains private cache state. Then, we profile the next 100 million instructions varying the available LLC ways. Based on these profiles we classify the benchmarks into three categories according to their LLC sensitivity. A benchmark has high LLC sensitivity (H)[5] if the speed-up with all LLC ways relative to a single way is greater than 1.75. If the speed-up is between 1.2 and 1.75, we classify the benchmark as having medium LLC sensitivity (M)[6]. The remaining benchmarks are classified as having low LLC sensitivity (L). Using this classification, we randomly generate 30 workloads with H-benchmarks, 15 workloads with M-benchmarks and 5 workloads with L-benchmarks for our 2-, 4- and 8-core CMPs (150 workloads in total)[7]. We explore the impact of workloads mixing H-, M- and L-benchmarks in Section VII-D.

For multi-programmed experiments, we simulate the workload until all benchmarks have committed 100 million instructions. Benchmarks are restarted when they reach the end of their instruction sample. We measure estimation accuracy by comparing these shared mode results to the corresponding private mode experiments. The shared mode instruction sample points are provided as input to the private mode experiments to ensure that measurements are based on the same instructions in both modes.

A shared mode estimate of a private mode value $\hat{\alpha}$ may differ from the actual private mode value α. To quantify this, we use the metrics absolute and relative error defined as $E^{\text{Abs}} = \hat{\alpha} - \alpha$ and $E^{\text{Rel}} = (\hat{\alpha} - \alpha)/\alpha$. We have configured all accounting techniques to estimate private mode performance every five million clock cycles since this is a common allocation period for resource management techniques (e.g., [8]). Thus, there will be multiple errors for each benchmark, and we use the *Root Mean Squared (RMS)* error metric to represent these errors with a single number:

$$\text{RMS} = \sqrt{1/n \times \sum_{i=1}^{n} E_i^2} \tag{8}$$

[5]*Apsi*, *facerec*, *galgel*, *ammp* and *art* from SPEC2000 and *omnetpp*, *lbm* and *sphinx3* from SPEC2006.

[6]*Equake*, *twolf*, *parser* and *vpr* from SPEC2000 and *gromacs*, *astar*, *bzip2* and *hmmer* from SPEC2006.

[7] We require that a benchmark is used at most once in a workload for the 2- and 4-core CMPs. For the 8-core CMP, we allow a benchmark to be used twice in the H and M workloads. The reason is that the H and M categories each contain 8 benchmarks which would result in a single workload if reuse was not allowed.

The key benefit of RMS is that it measures both bias and variability. In Equation 8, E can be either E^{Abs} or E^{Rel} and n is the number of estimates provided for a given benchmark. We use the arithmetic mean to represent multiple per-benchmark RMS errors with a single number.

VII. Results

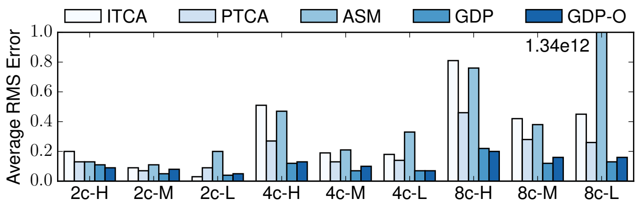
In this section, we present the results from our experimental evaluation. All techniques report performance estimates and re-evaluate allocations every 5 million clock cycles. We sample 32 sets in the LLC [8], and GDP and GDP-O have 32 PRB entries (see Section VII-D for evaluation).

A. Performance and Stall Time Estimation Accuracy

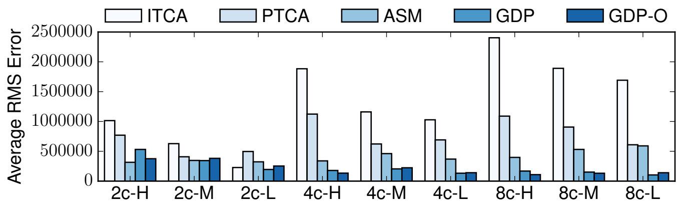
We compare the accuracy of GDP and GDP-O to *Inter-Task Conflict-Aware (ITCA)* accounting [4], PTCA [3] and ASM [15]. Since our system has an out-of-order memory controller, PTCA uses the private mode latency estimates provided by DIEF [21]. Figure 3 shows the average absolute RMS error of the private mode IPC and SMS-load-related stall cycle estimates for ITCA, PTCA, ASM, GDP and GDP-O. Figures 3a and 3b show the same trends for PTCA, GDP and GDP-O, and this illustrates that SMS-load stalls are the main source of interference-induced performance loss.

ITCA: ITCA is generally less accurate than the other techniques for workloads with more than negligible interference. The reason is that ITCA takes the shared mode stall cycles as the baseline and then subtracts the cycles where one of its selected conditions is true. Concretely, ITCA conditions are (i) a stalled rename stage with an inter-thread miss at the head of the ROB, (ii) all active MSHRs are inter-thread misses, or (iii) an empty ROB with an active inter-thread instruction miss. These conditions catch only a small part of the stall cycles due to interference, which results in conservative private mode estimates. This strategy works well when there is limited interference (i.e., the 2-core L-workloads), but gives larger errors than the other techniques in the remaining cases.

PTCA: Although PTCA is also an architecture-centric technique, it is significantly more accurate than ITCA in most cases. PTCA does not account the interference cycles that delay the SMS-load request belonging to the oldest instruction while the ROB is full. This illustrates that the choice of conditions is critically important. However, PTCA can also significantly underestimate private mode performance because they miss interference cycles due to stalls in other CPU structures. An example is *lbm* which in our simulation sample is in an inner loop with many floating-point adds and multiplies. This results in high pressure on the floating-point units and causes dispatch to stall due to a full issue queue. Thus, the ROB fills slowly and only a small part of the interference cycles are detected. GDP and GDP-O do not suffer from this problem since they use the critical path



(a) IPC estimate (Average absolute RMS error)



(b) SMS-load stall cycles (Average absolute RMS error)

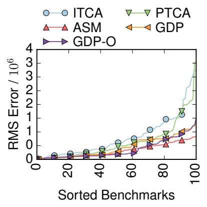
Figure 3: Average private mode prediction accuracy

length of the dependency graph to estimate private mode SMS-stalls.

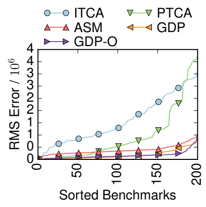
Another problem is that PTCA can severely overestimate private mode performance for benchmarks with significant SMS-load parallelism. This causes the accuracy degradation for the 2-core L-workloads in Figure 3. The main culprit is a workload where *libquantum* runs together with *tonto*. *Libquantum* is in a tight bandwidth-bound loop that sustains five concurrent SMS-loads, and each load enables committing a few instructions. One load causes a long stall and the remaining four cause short stalls because most of their latency was hidden by the long stall. PTCA treats these five stalls as isolated events. The long stall is estimated accurately since its length is dominated by interference. The requests causing the short stalls also experience abundant interference, and PTCA concludes that they did not occur in the private mode because the ROB is full. However, the stalls are present in the private mode since they are due to the loads being serialized in the memory controller. GDP and GDP-O use the dependency graph to determine that these load requests are serviced in parallel and therefore treat them as a single unit.

ASM: ASM's IPC estimates for the category L-workloads are highly inaccurate with an average RMS error of 1.34e12 for the 8-core CMP (Figure 3a). This error is mainly due to *applu* which has periods where practically all of its memory latency is due to interference-induced LLC misses, and the sample is dominated by stalls on these misses. Thus, almost all of the cycles in the period would not occur in the private mode. This leads ASM to estimate that the cache access rate would be very high because the substantial number of LLC accesses are divided by a small amount of clock cycles. This is not correct since *applu* does incur some latency accessing the other cache levels and on-chip interconnect. GDP and GDP-O handle this situation better than ASM because the performance model ensures that each source of processor stall cycles is handled independently.

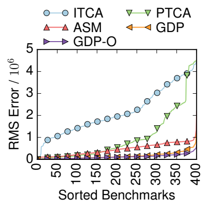
In Figure 3b, we combine ASM's slow-down estimates with the SMS-load-related stall cycle component of our performance model to estimate the private mode stall cycles. This improves accuracy significantly and leads to ASM being more accurate than ITCA and PTCA for most work-



(a) 2-core



(b) 4-core



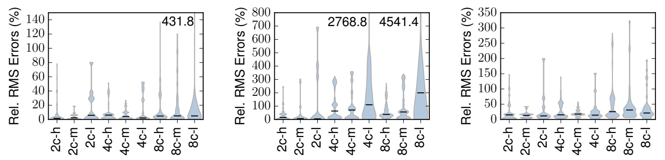
(c) 8-core

Figure 4: Absolute RMS error distribution for the SMS-load stall cycle predictions (y-axes in millions)

load categories and core counts. This indicates that ASM may be useful in combination with a performance model in situations where slowing down latency-sensitive processes is acceptable. GDP and GDP-O provide more accurate private mode stall cycle estimates than ASM for the 4- and 8-core CMPs and comparable accuracy for the 2-core CMP.

GDP and GDP-O: Figure 4 shows the private mode SMS-load-related stall cycle RMS error distributions for the 2-, 4- and 8-core CMPs. The distributions contain all workload categories. We generated these plots by retrieving the prediction errors for all workloads before we sorted the errors in ascending order. Overall, Figure 4 confirms the trends observed in Figure 3 and illustrates that GDP and GDP-O generally yield lower errors than ITCA, PTCA and ASM across our CMP configurations.

Perhaps counter-intuitively, Figure 4 shows that the accuracy benefit of GDP-O over GDP is limited. The reason is that applications are typically able to hide tens of cycles in the private mode while the average memory latencies are hundreds of cycles for memory-intensive applications. One exception is the 2-core H-workloads in Figure 4. Here, the applications are able to hide a significant part of the private mode memory latency which causes higher average error for GDP than GDP-O. Accurate private mode memory latency estimates is most important for the 4- and 8-core H-workloads, and this results in GDP-O only outperforming GDP by a small margin. These results indicate that the marginally simpler GDP technique can be used in situations where slightly lower accuracy is tolerable.



(a) CPL (b) Overlap (c) Latency

Figure 5: Relative RMS error distribution of GDP/GDP-O estimate components

B. GDP Component Prediction Accuracy

CPL Estimation Accuracy: The key underlying assumption of GDP and GDP-O is that CPL is very similar in the shared and private modes. Figure 5a shows a Violin plot[8] of the RMS error distribution of the CPL estimates generated with our runtime technique from Section IV compared to the same algorithms running with unlimited buffer space in the private mode. Thus, Figure 5a covers all sources of CPL inaccuracies, including our PRB management policy. The main observation is that the relative RMS error is less than 10% for the majority of benchmarks. Thus, the assumption that CPL is very similar in the shared and private modes holds in most cases.

However, all workload categories have clusters of benchmarks with higher CPL errors. Higher relative errors can occur when the critical path is short. In this case, even a relatively small absolute deviation can cause a large relative error. A notable example is *wrf* in the 8-core L-workloads where the relative error is 431.8% while the absolute RMS error is only 7.8. However, GDP's performance estimate error is only 5.0%. The reason is that *wrf* is compute-bound which means that the memory-related stall cycles account for a small fraction of the total stall cycles. The same problem occurs with *facerec* in the H-workloads, and this causes the highest errors for this data series. *Facerec* is alternating between compute-bound and memory-bound phases, and the CPL estimates are accurate when *facerec* is memory-bound.

CPL differences are also caused by private mode shared cache hits that enable the processor to continuously commit instructions even when a later load instruction depends on an earlier load. If these cache accesses become interference misses in the shared mode, the shape of the dependency graph will change since the dependencies become observable in the memory system. Figure 5a shows that the conditions that cause CPL changes in memory-bound phases are relatively rare. In fact, we have not observed a single case where inaccurate CPL estimates can be blamed for significant private mode performance estimation errors.

Overlap Estimation Accuracy: Figure 5b contains a Violin plot showing the RMS error distribution of GDP-O's overlap estimator. It shows that our overlap estimator can be inaccurate, but that it performs reasonably as long as the overlap behavior is similar in the shared and private modes (i.e., on the 2-core and the 4- and 8-core H- and M-workloads). Our estimator is inaccurate for the 4- and 8-core L-workloads. However, this inaccuracy does not adversely affect the performance estimates because SMS-load-related stall cycles have a limited impact on accuracy for compute-bound applications. For example, the worst-case error of 4541.4% is due to the compute-bound *h264ref* benchmark, but GDP-O's IPC estimation error is only 2.2%. Thus, we conclude that our simple overlap estimator is sufficient, and we avoid further complicating the design.

Memory Latency Estimation Accuracy: Figure 5c uses a Violin plot to show the RMS error distributions of DIEF's private mode latency estimates. The latency estimates are relatively accurate for the majority of the workloads with a maximum median RMS error of 31%. However, there are cases where larger errors occur, and the root cause of these errors are inaccurate memory bus queuing latency estimates. For the L-workloads, the queuing latencies are relatively small which results in significant relative errors even if the estimates are only off by a few tens of clock cycles. In fact, the worst observed mean error is 140.8 clock cycles. Again, the errors do not adversely affect the performance estimates because the L-workloads are mostly compute-bound.

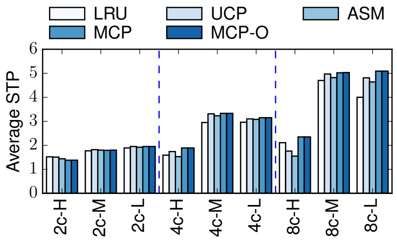
There are cases where DIEF's estimates are inaccurate. However, these do not result in unmanageable performance prediction errors due to a phenomenon similar to Amdahl's law. The key insight is that the memory latency estimate is one of many components of the performance model. Thus, the performance estimate impact of a large error in one component is reduced if the errors of the other components are lower. The results in Section VII-A shows that DIEF is sufficiently accurate to enable GDP and GDP-O outperform prior work. However, there might be use cases where more accurate private mode latency estimates are helpful.

C. Shared Cache Management Case Study

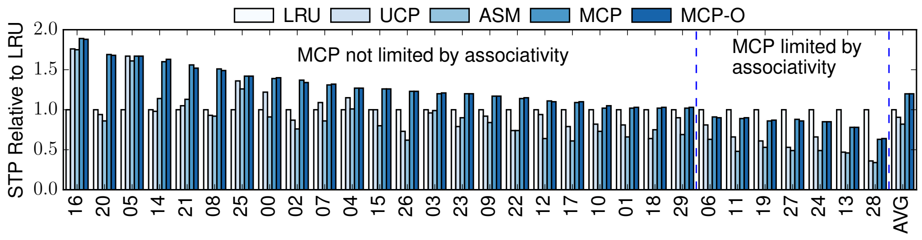
In this section, we evaluate our *Model-based Cache Partitioning (MCP)* scheme to illustrate the benefits of using private mode performance estimates for dynamic memory system resource management. We compare to the conventional LRU replacement policy, *Utility-based Cache Partitioning (UCP)* [8] and ASM-driven cache partitioning [15]. We evaluate two versions of MCP: MCP and MCP-O. MCP uses GDP to provide private mode performance estimates and MCP-O uses GDP-O. We use the throughput-oriented *System Throughput (STP)* [18] performance metric. STP is the sum of the private to shared mode CPI slowdowns across all cores in the CMP (i.e., $\text{STP} = \sum_{i=0}^{n} \pi_i / \text{CPI}_i$).

Figure 6a shows the average STP for LRU, UCP, ASM, MCP and MCP-O across all workload categories and core

[8]A Violin plot uses kernel density estimation to illustrate the distribution of a series of discrete data points. The horizontal line shows the median of each data series.



(a) Average system throughput



(b) System throughput for the 8-core category H-workloads relative to LRU

Figure 6: System throughput with LLC partitioning

counts. The main observation is that accurate private mode performance estimates enable MCP and MCP-O to outperform the other policies on the 4- and 8-core CMPs. Resource contention is limited on the 2-core CMP, and this results in all techniques performing similarly. For the 8-core H-workloads, MCP improves average STP by 11%, 34% and 52% compared to LRU, UCP and ASM, respectively. MCP also provides a significant STP improvement for the 4-core H-workloads (i.e., 19%, 9% and 24%). These results indicate that accurate private mode performance estimates are most useful when there is significant competition for LLC capacity. However, MCP and MCP-O are also the top performers for the other workload types on the 4- and 8-core CMPs, but the performance difference is smaller.

Figure 6a also shows that LRU outperforms UCP and ASM for the 8-core H-workloads. To explain this result, we have included Figure 6b which show the STP for all H-workloads on the 8-core CMP. Overall, Figure 6b shows that MCP and MCP-O provide significant STP improvements relative to LRU (up to 88%). UCP and ASM are able to approach the performance of MCP and MCP-O for some workloads (e.g., 16, 5 and 25). However, there are also a number of workloads where MCP and MCP-O significantly improve STP where UCP and ASM perform similarly to LRU (e.g., 20, 14, 21 and 8). This indicates that highly accurate private mode performance estimates enable MCP and MCP-O to find good cache allocations in situations where ASM cannot.

LRU outperforms UCP and ASM on average because way-partitioning allocates LLC capacity in a coarse-grained manner. This has led researchers to propose management techniques such as Vantage [31] that target high-associativity cache structures such as z-caches [32] and molecular caches [33]. Such cache structures add non-negligible complexity compared to conventional set-associative caches which implies that they are challenging to design, verify and deploy in commercial processors [34]. Figure 6b shows that MCP and MCP-O have significantly smaller STP regressions relative to LRU than UCP and ASM for the associativity-limited workloads. This indicates that high-accuracy private mode performance estimates help alleviate a well-known limitation of way-partitioning-based LLC management schemes.

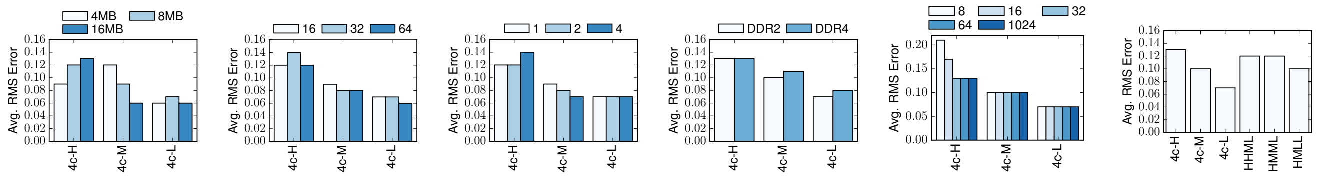
D. Sensitivity Analysis

Figure 7 shows GDP-O's accuracy when changing architectural parameters and configuration settings as well as when running mixes of H, M and L-benchmarks. The main observation is that GDP-O achieves high accuracy for almost all configurations. For the M-workloads, error either decreases with more resources or is unaffected by parameter changes. The reason is that more resources result in less contention which leads to smaller differences between the shared and private modes. In other words, more resources makes the estimation problem easier. On the architectures used in this work, the L-workloads do not exhibit significant contention and have relatively stable accuracy.

LLC Parameters: Figure 7a shows that RMS error increases slightly with LLC size for the H-workloads. This is primarily due to *facerec*. With the 8 MB and 16 MB LLCs, *facerec* is able to cache one of its working sets. This results in performance mainly being limited by the ability of the memory system to service loads. For the 4 MB LLC, *facerec* is bound by the ability of the memory system to service stores, and interference causes the store buffer to block. This is a significantly easier estimation problem. Figure 7b shows that the accuracy of GDP-O is relatively stable across different LLC associativities.

DRAM System: Figure 7c shows the accuracy impact of changing the number of DDR2-channels, and Figure 7d explores the impact of changing the single-channel interface from DDR2-800 to DDR4-2666. Overall, GDP-O is robust to changes in memory bandwidth. The slightly higher errors for the 4-channel DDR2 H-workloads and the DDR4 M and L-workloads are caused by increased private mode latency estimation errors. Since future computer systems are expected to be bandwidth-limited [35], it is important that accounting techniques are accurate for resource-constrained configurations [36].

PRB Entries: Figure 7e shows the accuracy impact of changing the number of PRB entries for GDP-O. Intuitively, the PRB needs enough entries to track the dependencies of



(a) LLC size (b) LLC associativity (c) DDR2 channels (d) DRAM interface (e) PRB size (f) Mixed workloads

Figure 7: Average absolute RMS error of GDP-O IPC estimates for the 4-core CMP

the in-flight load requests. For the M- and L-workloads, 8 PRB entries are sufficient to reach the accuracy potential. For the H-workloads, a 32-entry PRB is necessary because the highly LLC-sensitive benchmarks exhibit significant MLP. Increasing the number of PRB entries beyond 32 does not improve accuracy because the amount of MLP is bounded by the CPU's 32-entry load/store queue.

Mixed Workloads: We also explore the sensitivity of GDP-O to workloads that contain benchmarks with different characteristics. To achieve this, we generate three mixed workload types that contain two H, M or L-benchmarks and one benchmark from each of the two remaining categories. We identify the mixes with the type of each benchmark, and randomly generate 10 workloads for each mix. For example, the HHML workload type contains two H-type benchmarks, one M-benchmark and one L-benchmark. Figure 7f shows GDP-O's average RMS error for the HHML, HMML and HMLL workload mixes and compares it to the results of the H, M and L-workloads. Overall, the error of GDP-O is relatively insensitive to how the workloads are constructed. However, we observe somewhat higher errors for the workloads that contain H-benchmarks. The reason is that the H-benchmarks create significant interference which makes the private mode performance estimation problem more challenging.

VIII. Related Work

Performance Accounting: A number of transparent accounting systems exist. PTCA [3] assumes that performance loss due to memory system interference occurs when the ROB is full and the oldest SMS-load request is delayed by cache or memory bus interference. ITCA [4, 17] uses a similar heuristic. The differences are that ITCA defines a processor stall using a stalled register rename stage and does not account for interference in the off-chip memory bus for intra-task shared cache misses. Furthermore, ITCA defines cycles where all active MSHRs contain inter-thread misses and cycles where the ROB is empty due to an inter-thread instruction miss as interference. We compare to both PTCA and ITCA in our experiments and show that GDP and GDP-O are significantly more accurate. In addition, Zhou et al. [7] propose a performance estimation technique for the shared cache, but do not consider memory bus sharing effects.

Other researchers have proposed invasive accounting systems which estimate private mode performance by periodically running an application in isolation or with high priority. Cazorla et al. [16] estimate private mode performance by periodically running each selected process in isolation. MISE [15] is a memory bus scheduler that estimates private mode performance by periodically running each process with the highest priority in the memory controller. ASM [14] extends the MISE model to account for LLC interference. We compare to ASM and show that GDP and GDP-O are significantly more accurate. Invasive accounting systems may also slow down latency-sensitive processes. In contrast, GDP can estimate private mode performance without any performance overhead for the running applications.

Accounting schemes for SMT processors have also been proposed (e.g., [37, 38]). Here, the focus is on handling interference in the processor core resources and the per-core memory system. Thus, they are complementary to this work.

Accounting Heuristics: In addition to the research that targets private mode performance estimation, there is a significant body of work that detects interference cycles but do not detail how this additional latency impacts performance. Ebrahimi et al. [39] estimate the excess cycles due to interference using a pollution filter to detect cache interference and detecting bus conflicts, DRAM bank conflicts and row-buffer interference. The Cache Scouts [40] mechanism collects information on cache interference for use by the OS or virtual machine monitor. Mutlu et al. [10] detect memory bus interference by using the shared mode memory bus queue and bank state to estimate the additional shared mode memory latency. In fact, most cache management schemes that rely on ATDs fall into this category (e.g., [8]) since they commonly use private mode shared cache misses as a proxy for private mode performance. These works all validate their interference estimates by showing that their proposed interference-aware schemes perform well. We argue that estimation techniques should be considered first-class citizens of a resource management system and thoroughly evaluated with respect to their accuracy.

LLC Partitioning: A large body of research targets shared LLC capacity management (e.g., [7, 8, 9, 23, 31, 34]). Of these, MCFQ [41] and ASM [14] are most closely related to this work since they determine per-process LLC quotas by combining private mode miss curves obtained with ATDs and a performance model. MCFQ does not take private mode performance into account. This may lead to prioritizing high IPC processes over low IPC processes which may not be in accordance with high-level system management objectives. We compare to ASM in our evaluation and show that our MCP scheme provides higher throughput.

In addition, a large body of work use ATDs and miss-minimization within their shared LLC partitioning schemes [8, 23, 31, 34, 42, 43, 44, 45]. These are complementary to this work because using MCP would enable them to select partitions based on system performance rather than LLC misses. Another common optimization target is QoS [7, 9, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55]. These works are also complementary because MCP can enable specifying QoS targets in terms of performance rather than resource requirements. Finally, researchers have proposed to partition the LLC to improve energy efficiency by disabling cache ways that are of little utility to the current workload [56, 57, 58]. MCP enables making the decision of disabling ways using performance estimates rather than LLC misses.

Dynamic Voltage and Frequency Scaling (DVFS): Prior work has used *leading loads* [59, 60, 61, 62] to estimate the performance impact of changing processor core frequencies. Informally, a leading load is the first load in a load burst. Counting the cycles a leading load is pending estimates the memory-related stall cycles, and these cycles are relatively unaffected by core frequency changes. Stall cycle partitioning is also an important accounting system component, and leading loads is similar to the heuristics employed in architecture-centric accounting (e.g., PTCA [3]). However, the leading load mechanisms cannot be directly applied to performance accounting since they lack support for estimating how memory system interference affects stall cycles.

Instruction Criticality: Researchers have proposed techniques for detecting and exploiting instruction criticality. Of these, Fields et al. [63] is most similar to our work since they use a CPU-internal instruction criticality detector. In contrast, we propose to track dependencies within the memory system which has several advantages for accounting systems. Concretely, we avoid high-frequency interactions with internal CPU resources, avoid tracking dependencies among non-load instructions and avoid the increased verification costs associated with complicating the CPU design. The cost of exact criticality and dependency tracking has motivated researchers to develop heuristic approaches [64, 65, 66]. Such heuristics are similar to the heuristics used in architecture-centric accounting which we have shown to be less accurate than dataflow accounting.

IX. Conclusion

Shared memory system resources lead to destructive interference between co-scheduled applications and reduced performance predictability. This problem can be alleviated by interference-aware OS schedulers or thread-aware memory system management policies. In both cases, accurate estimates of private mode performance is a great asset. In this work, we have presented the GDP and GDP-O estimation techniques. They explicitly model the relationship between SMS-load latency and CPU stall cycles, are very accurate and can be implemented with only a small storage overhead. We also presented the MCP LLC partitioning scheme to illustrate how highly accurate private mode performance estimates can be used in a memory system resource management policy. GDP enables MCP to dynamically estimate system throughput [18] which MCP uses to partition the shared cache. For the 4- and 8-core CMPs, MCP is able to improve average throughput by 11.9% and 20.8% compared to ASM-driven cache partitioning [14].

Acknowledgment

We would like to thank the researchers at NTNU's Computer Architecture Lab and the anonymous reviewers for their insightful comments on the research presented in this work. Lieven Eeckhout is supported through FWO grants no. G.0434.16N and G.0144.17N, and European Research Council (ERC) Advanced Grant agreement no. 741097. The simulations were performed on resources provided by UNINETT Sigma2 – the National Infrastructure for High Performance Computing and Data Storage in Norway.

References

[1] K. J. Nesbit *et al.*, "Multicore resource management," *IEEE micro*, vol. 28, no. 3, 2008.

[2] M. Armbrust *et al.*, "Above the clouds: A Berkeley view of cloud computing," University of California at Berkeley, Tech. Rep., 2009.

[3] K. Du Bois *et al.*, "Per-thread cycle accounting in multicore processors," *ACM Trans. on Arch. and Code Optimization*, 2013.

[4] C. Luque *et al.*, "CPU accounting for multicore processors," *IEEE Transactions on Computers*, vol. 61, no. 2, 2012.

[5] R. Das *et al.*, "Application-to-core mapping policies to reduce memory system interference in multi-core systems," in *Int. Symp. on High Performance Computer Architecture (HPCA)*, 2013.

[6] A. Jaleel *et al.*, "CRUISE: Cache replacement and utility-aware scheduling," in *Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012.

[7] X. Zhou *et al.*, "Cache sharing management for performance fairness in chip multiprocessors," in *Int. Conf. on Parallel Architectures and Compilation Techniques (PACT)*, 2009.

[8] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *Int. Symp. on Microarchitecture (MICRO)*, 2006.

[9] R. Wang and L. Chen, "Futility scaling: High-associativity cache partitioning," in *Int. Symp. on Microarchitecture (MICRO)*, 2014.

[10] O. Mutlu and T. Moscibroda, "Stall-time fair memory access scheduling for chip multiprocessors," in *International Symposium on Microarchitecture (MICRO)*, 2007.

[11] ——, "Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems," in *Int. Symp. on Computer Architecture (ISCA)*, 2008.

[12] Y. Kim *et al.*, "Thread cluster memory scheduling: Exploiting differences in memory access behavior," in *International Symposium on Microarchitecture (MICRO)*, 2010.

[13] M. Xie *et al.*, "Improving system throughput and fairness simultaneously in shared memory CMP systems via dynamic bank partitioning," in *Int. Symp. on High Perf. Computer Architecture (HPCA)*, 2014.

[14] L. Subramanian *et al.*, "The application slowdown model: Quantifying and controlling the impact of inter-application interference at shared caches and main memory," in *International Symposium on Microarchitecture (MICRO)*, 2015.

[15] ——, "MISE: Providing performance predictability and improving fairness in shared main memory systems," in *Int. Symp. on High Performance Computer Architecture (HPCA)*, 2013.

[16] F. J. Cazorla *et al.*, "Predictable performance in SMT processors: Synergy between the OS and SMTs," *IEEE Transactions on Computers*, vol. 55, no. 7, 2006.

[17] C. Luque *et al.*, "ITCA: Inter-task conflict-aware CPU accounting for CMPs," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2009.

[18] S. Eyerman and L. Eeckhout, "System-level performance metrics for multiprogram workloads," *IEEE Micro*, vol. 28, no. 3, 2008.

[19] A. B. Kahn, "Topological sorting of large networks," *Communications of the ACM*, vol. 5, no. 11, 1962.

[20] T. S. Karkhanis and J. E. Smith, "A first-order superscalar processor model," in *Int. Symp. on Computer Architecture (ISCA)*, 2004.

[21] M. Jahre *et al.*, "DIEF: An accurate interference feedback mechanism for chip multiprocessor memory systems," in *Int. Conf. on High-Performance Embedded Arch. and Compilers (HiPEAC)*, 2010.

[22] M. K. Qureshi *et al.*, "A case for MLP-aware cache replacement," in *International Symposium on Computer Architecture (ISCA)*, 2006.

[23] Y. Xie and G. H. Loh, "PIPP: Promotion/insertion pseudo-partitioning of multi-core shared caches," in *International Symposium on Computer Architecture (ISCA)*, 2009.

[24] S. Rixner *et al.*, "Memory access scheduling," in *International Symposium on Computer Architecture (ISCA)*, 2000.

[25] N. L. Binkert *et al.*, "The M5 simulator: Modeling networked systems," *IEEE Micro*, vol. 26, no. 4, 2006.

[26] J. Casazza, "Intel core i7-800 processor series and the Intel core i5-700 processor series based on Intel microarchitecture (Nehalem)," 2009, Published: White paper, Intel Corp.

[27] JEDEC, *DDR2 SDRAM specification*, 2009.

[28] ——, *DDR4 SDRAM standard*, 2017.

[29] SPEC, "SPEC CPU2000 web page," 2007. [Online]. Available: http://www.spec.org/cpu2000/

[30] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *SIGARCH Computer Architecture News*, vol. 34, no. 4, 2006.

[31] D. Sanchez and C. Kozyrakis, "Vantage: Scalable and efficient fine-grain cache partitioning," in *International Symposium on Computer Architecture (ISCA)*, 2011.

[32] ——, "The ZCache: Decoupling ways and associativity," in *International Symposium on Microarchitecture (MICRO)*, 2010.

[33] K. Varadarajan *et al.*, "Molecular Caches: A caching structure for dynamic creation of application-specific heterogeneous cache regions," in *Int. Symp. on Microarchitecture (MICRO)*, 2006.

[34] R. Manikantan *et al.*, "Probabilistic shared cache management (PriSM)," in *Int. Symp. on Computer Architecture (ISCA)*, 2012.

[35] J. Huh *et al.*, "Exploring the design space of future CMPs," in *Int. Conf. on Parallel Arch. and Compilation Techniques (PACT)*, 2001.

[36] M. Jahre *et al.*, "A quantitative study of memory system interference in chip multiprocessor architectures," in *International Conference on High Performance Computing and Communications (HPCC)*, 2009.

[37] S. Eyerman and L. Eeckhout, "Per-thread cycle accounting in SMT processors," in *Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2009.

[38] C. Luque *et al.*, "Fair CPU time accounting in CMP+SMT processors," *ACM Trans. on Arch. and Code Opt.*, vol. 9, no. 4, 2013.

[39] E. Ebrahimi *et al.*, "Fairness via source throttling: A configurable and high-performance fairness substrate for multi-core memory systems," in *Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2010.

[40] L. Zhao *et al.*, "CacheScouts: Fine-grain monitoring of shared caches in CMP platforms," in *Int. Conf. on Parallel Architecture and Compilation Techniques (PACT)*, 2007.

[41] D. Kaseridis *et al.*, "Cache friendliness-aware management of shared last-level caches for high performance multi-core systems," *IEEE Transactions on Computers*, vol. 63, no. 4, 2014.

[42] H. Dybdahl *et al.*, "A cache-partitioning aware replacement policy for chip multiprocessors," in *International Conference on High Performance Computing (HiPC)*, 2006.

[43] Y. Xie and G. H. Loh, "Scalable shared-cache management by containing thrashing workloads," in *Int. Conf. on High-Performance Embedded Architectures and Compilers (HiPEAC)*, 2010.

[44] D. Zhan *et al.*, "CLU: Co-optimizing locality and utility in thread-aware capacity management for shared last level caches," *IEEE Transactions on Computers*, vol. 63, no. 7, 2014.

[45] J. J. K. Park *et al.*, "Fine grain cache partitioning using per-instruction working blocks," in *International Conference on Parallel Architecture and Compilation Techniques (PACT)*, 2015.

[46] R. Iyer, "CQoS: A framework for enabling QoS in shared caches of CMP platforms," in *Proceedings of the 18th Annual International Conference on Supercomputing (ICS)*, 2004.

[47] P. Petoumenos *et al.*, "STATSHARE: A statistical model for managing cache sharing via decay," in *Second Annual Workshop on Modeling, Benchmarking and Simulation (MoBS)*, 2006.

[48] F. Guo *et al.*, "A framework for providing quality of service in chip multi-processors," in *International Symposium on Microarchitecture (MICRO)*, 2007.

[49] K. J. Nesbit *et al.*, "Virtual private caches," in *International Symposium on Computer Architecture (ISCA)*, 2007.

[50] N. Rafique *et al.*, "Architectural support for operating system-driven CMP cache management," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2006.

[51] J. Chang and G. S. Sohi, "Cooperative cache partitioning for chip multiprocessors," in *Int. Conf. on Supercomputing (ICS)*, 2007.

[52] S. Srikantaiah *et al.*, "SHARP control: controlled shared cache management in chip multiprocessors," in *International Symposium on Microarchitecture (MICRO)*, 2009.

[53] M. Moreto *et al.*, "FlexDCP: a QoS framework for CMP architectures," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 2, 2009.

[54] A. Herdrich *et al.*, "Cache QoS: From concept to reality in the Intel Xeon processor E5-2600 v3 product family," in *Int. Symp. on High Performance Computer Architecture (HPCA)*, 2016.

[55] H. Kasture and D. Sanchez, "Ubik: Efficient cache sharing with strict QoS for latency-critical workloads," in *Int. Conf. on Arch. Support for Programming Languages and Operating Systems (ASPLOS)*, 2014.

[56] K. T. Sundararajan *et al.*, "Cooperative partitioning: Energy-efficient cache partitioning for high-performance CMPs," in *Int. Symp. on High-Performance Computer Architecture (HPCA)*, 2012.

[57] ——, "RECAP: Region-aware cache partitioning," in *International Conference on Computer Design (ICCD)*, 2013.

[58] H. Cook *et al.*, "A hardware evaluation of cache partitioning to improve utilization and energy-efficiency while preserving responsiveness," in *Int. Symp. on Computer Architecture (ISCA)*, 2013.

[59] B. Rountree *et al.*, "Practical performance prediction under Dynamic Voltage Frequency Scaling," in *International Green Computing Conference and Workshops*, 2011.

[60] S. Eyerman and L. Eeckhout, "A counter architecture for online DVFS profitability estimation," *IEEE Transactions on Computers*, vol. 59, no. 11, 2010.

[61] G. Keramidas *et al.*, "Interval-based models for run-time DVFS orchestration in superscalar processors," in *International Conference on Computing Frontiers (CF)*, 2010.

[62] B. Su *et al.*, "PPEP: Online performance, power, and energy prediction framework and DVFS space exploration," in *Int. Symp. on Microarchitecture (MICRO)*, 2014.

[63] B. Fields *et al.*, "Focusing processor policies via critical-path prediction," in *Int. Symp. on Computer Architecture (ISCA)*, 2001.

[64] S. Subramaniam *et al.*, "Criticality-based optimizations for efficient load processing," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2009.

[65] S. Ghose *et al.*, "Improving memory scheduling via processor-side load criticality information," in *International Symposium on Computer Architecture (ISCA)*, 2013.

[66] M. Hashemi *et al.*, "Accelerating dependent cache misses with an enhanced memory controller," in *International Symposium on Computer Architecture (ISCA)*, 2016.