PROFET: Modeling System Performance and Energy Without Simulating the CPU

MILAN RADULOVIC, Barcelona Supercomputing Center (BSC) & Universitat Politècnica de Catalunya (UPC)
ROMMEL SÁNCHEZ VERDEJO, Barcelona Supercomputing Center (BSC) & Universitat Politècnica de Catalunya (UPC)
PAUL CARPENTER, Barcelona Supercomputing Center (BSC)
PETAR RADOJKOVIĆ, Barcelona Supercomputing Center (BSC)
BRUCE JACOB, University of Maryland
EDUARD AYGUADÉ, Barcelona Supercomputing Center (BSC) & Universitat Politècnica de Catalunya (UPC)

The approaching end of DRAM scaling and expansion of emerging memory technologies is motivating a lot of research in future memory systems. Novel memory systems are typically explored by hardware simulators that are slow and often have a simplified or obsolete abstraction of the CPU. This study presents PROFET, an analytical model that predicts how an application's performance and energy consumption changes when it is executed on different memory systems. The model is based on instrumentation of an application execution on actual hardware, so it already takes into account CPU microarchitectural details such as the data prefetcher and out-of-order engine. PROFET is evaluated on two real platforms: Sandy Bridge-EP E5-2670 and Knights Landing Xeon Phi platforms with various memory configurations. The evaluation results show that PROFET's predictions are accurate, typically with only 2% difference from the values measured on actual hardware. We release the PROFET source code and all input data required for memory system and application profiling. The released package can be seamlessly installed and used on high-end Intel platforms.

CCS Concepts: • **Computing methodologies** → **Model development and analysis**; • **Hardware** → **Dynamic memory**.

Additional Key Words and Phrases: Memory bandwidth; Memory access latency; DRAM; MCDRAM; Performance; Power; Energy; Modeling

ACM Reference Format:
Milan Radulovic, Rommel Sánchez Verdejo, Paul Carpenter, Petar Radojković, Bruce Jacob, and Eduard Ayguadé. 2019. PROFET: Modeling System Performance and Energy Without Simulating the CPU. *Proc. ACM Meas. Anal. Comput. Syst.* 3, 2, Article 34 (June 2019), 33 pages. https://doi.org/10.1145/3326149

Authors' addresses: Milan Radulovic, Barcelona Supercomputing Center (BSC) & Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, milan.radulovic@bsc.es; Rommel Sánchez Verdejo, Barcelona Supercomputing Center (BSC) & Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, rommel.sanchez@bsc.es; Paul Carpenter, Barcelona Supercomputing Center (BSC), Barcelona, Spain, paul.carpenter@bsc.es; Petar Radojković, Barcelona Supercomputing Center (BSC), Barcelona, Spain, petar.radojkovic@bsc.es; Bruce Jacob, University of Maryland, College Park, Maryland, USA, blj@umd.edu; Eduard Ayguadé, Barcelona Supercomputing Center (BSC) & Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, eduard@ac.upc.edu.

2476-1249/2019/6-ART34 $15.00
https://doi.org/10.1145/3326149

1 INTRODUCTION

The memory system is a major contributor to the deployment and operational costs of a large-scale high-performance computing (HPC) cluster [24, 35, 38], and in terms of system performance it is one of the most critical aspects of the system's design [20, 41]. For decades, most server and HPC cluster memory systems have been based on DRAM DIMMs. However, it is becoming questionable whether DRAM DIMMs will continue to scale and meet the industry's demand for high performance and high capacity memory. Significant effort is therefore being invested into the research and development of future memory systems.

Application performance on novel memory systems is typically estimated using a hardware simulator. The simulation is, however, time consuming, which limits the number of design options that can be explored within a practical length of time. Also, although memory simulators are typically well validated [23, 32], current CPU simulators have various shortcomings, such as simplified out-of-order execution, an obsolete data prefetcher and a lack of virtual-to-physical memory translation, all of which can make a huge difference between the simulated and actual memory system, in terms of behavior and performance.

This study proposes PROFET (PROFiling-based EsTimation of performance and energy), an analytical model that predicts how an application's performance, power and energy consumption would change when it is executed on a new memory system. The method is based on instrumentation of an application running on actual hardware, so it already takes account of CPU microarchitectural details such as the real (and not publicly disclosed) data prefetcher and out-of-order engine. Therefore, it can be used to model various platforms as long as they support the required application profiling. PROFET was initially developed for the Sandy Bridge platform, and later we evaluated it for the Knights Landing (KNL) server. Adjustment of the PROFET model to the KNL system was trivial, as it required changes to only a few hardware parameters, such as, for example the reorder buffer size.

We evaluated PROFET on two actual platforms: Sandy Bridge-EP E5-2670 with four DRAM configurations DDR3-800/1066/1333/1600, and Knights Landing Xeon Phi with DDR4 and 3D-stacked MCDRAM. The evaluation results show that PROFET's predictions are very accurate: the average difference from the performance, power and energy measured on the actual hardware is only 2%, 1.1% and 1.7%, respectively. We also compare PROFET's performance predictions with simulation results for the Sandy Bridge-EP E5-2670 system with ZSim [33, 40] and DRAMSim2 [32], and PROFET shows significantly better accuracy over the simulator. PROFET is also faster than the hardware simulators by *three orders of magnitude*, so it can be used to analyze production HPC applications, on arbitrarily sized systems.

We release the PROFET source code as open source [31]. The release includes all inputs and outputs and evaluation results for the case study that is used in the rest of this paper. The package includes the memory system profiles, CPU parameters, application profiles and memory power parameters, as well as the power, performance and energy outputs from PROFET and the measurements on the baseline and target platforms. The released PROFET model is ready to be used on high-end Intel platforms, and we would encourage the community to use it, adapt it to other platforms, and share their own evaluations.

2 PROFET OVERVIEW

This section summarizes the main idea behind PROFET's analytical models and it describes the inputs and outputs to PROFET.

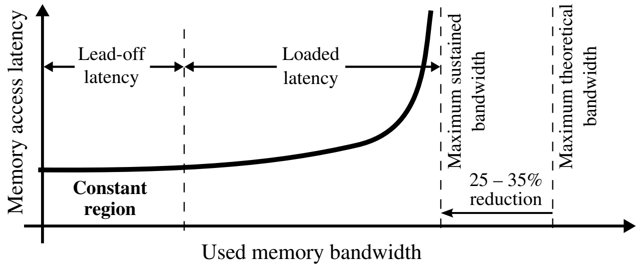


Fig. 1. Bandwidth–latency curve showing how the memory access latency depends on the used memory bandwidth. It is critical to distinguish between the *lead-off* and *loaded* memory access latency regions [20].

2.1 Background: Memory bandwidth and latency

The memory access latency and used bandwidth are often described as independent concepts, but they are in fact inherently interrelated [20]. We start by clarifying what is meant by the *lead-off* and *loaded* memory access latencies, and then we address the connection between memory access latency and used memory bandwidth. **Lead-off memory access latency** corresponds to the single-access read latency in an idle system. This latency includes the time spent in the CPU load/store queues, cache memory, memory controller, memory channel and main memory. **Loaded memory access latency** corresponds to the read latency in a loaded system. In addition to all timings included in the lead-off latency, the loaded memory latency includes shared-resource contention among concurrent memory requests. As illustrated in Figure 1, the loaded memory access latency increases (non-linearly) with the used bandwidth, due to increasing contention among concurrent memory requests. It is critical to distinguish between the lead-off and loaded latencies because the difference between them can be on the order of hundreds of nanoseconds.

2.2 The idea: Moving between memory curves

The main idea of this paper is that we can understand the effect of changing the memory system by understanding how the application moves from one bandwidth–latency curve to another. We illustrate this idea using the DDR4 and MCDRAM memories on Intel's Knights Landing platform. This platform has two memory systems, so there are two bandwidth–latency curves, shown together on the same plot in Figure 2.[1] When used bandwidth is high, as seen towards the right of the figure,

[1] Figure 2 shows a simplified bandwidth–latency curve, as discussed in Section 2.1. Detailed curves are given in Figure 4 in Section 3.

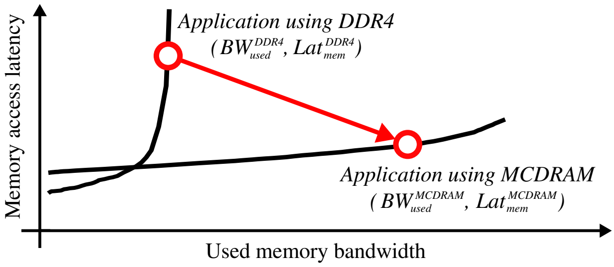


Fig. 2. High-level view of the transition from DDR4 to high-bandwidth MCDRAM memory on the KNL platform.

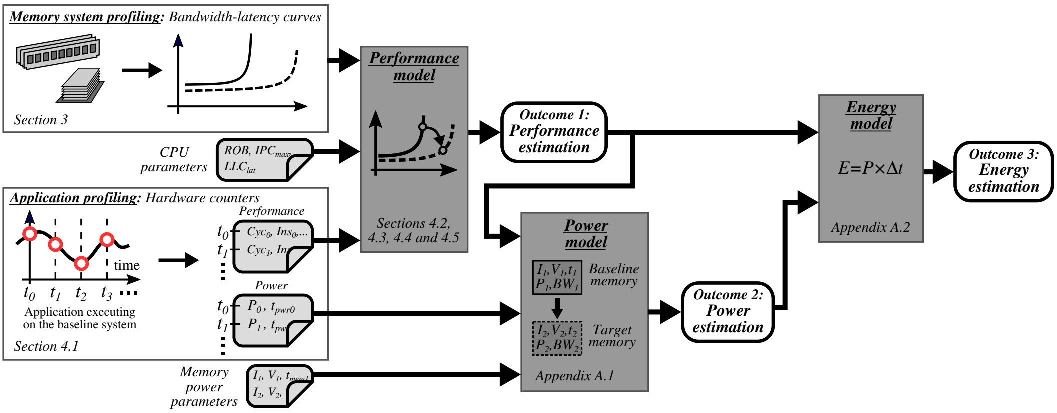

Memory system profiling: Bandwidth-latency curves
Section 3
CPU parameters
ROB, IPC$_{max}$, LLC$_{lat}$
Application profiling: Hardware counters
t_0 t_1 t_2 t_3
time
Application executing on the baseline system
Section 4.1
Performance
t_0 Cyc$_0$, Ins$_0$,...
t_1 Cyc$_1$, In
Power
t_0 P_0, t_{pwr0}
t_1 P_1, t_{pw}
Memory power parameters
I_1, V_1, t_{mem1}
I_2, V_2,
Performance model
Sections 4.2, 4.3, 4.4 and 4.5
Outcome 1: Performance estimation
Power model
I_1,V_1,t_1 P_1,BW_1 Baseline memory
I_2,V_2,t_2 P_2,BW_2 Target memory
Appendix A.1
Outcome 2: Power estimation
Energy model
$E=P\times\Delta t$
Appendix A.2
Outcome 3: Energy estimation

Fig. 3. Diagram of the whole process of performance, power and energy estimation. The cross-references indicate which section describes which part of the estimation process.

MCDRAM is clearly better. In contrast, when used bandwidth is low, as seen towards the left, DDR4 has lower latency due to its lower lead-off latency.

When an application (or application phase) executes on the DDR4 main memory, it will be positioned at some point on the DDR4 curve; e.g. $(BW_{used}^{DDR4},\ lat_{\mathrm{mem}}^{DDR4})$ illustrated in Figure 2. Analogously, when the same application is executed on the MCDRAM memory, it will be positioned at some point on the MCDRAM curve, e.g. $(BW_{used}^{MCDRAM},\ lat_{\mathrm{mem}}^{MCDRAM})$. We see that the application in Figure 2 benefits from running on the MCDRAM through a lower memory latency (MCDRAM point is lower) and a higher used bandwidth (MCDRAM point is to the right). This idea, of moving between bandwidth–latency curves, is central to the PROFET performance, power and energy models presented in this paper.

2.3 PROFET inputs

Figure 3 gives a high-level overview of the whole process of performance, power and energy estimation. The inputs to PROFET, shown towards the left of the figure, are the bandwidth–latency curves, measured for the *baseline memory system* and the *target memory system*, parameters for the CPU (which is the same for both memory systems), as well as the application profiles on the baseline memory system. These inputs can all be easily obtained on mainstream platforms and many emerging platforms. The outputs from PROFET will be the predicted performance, power and energy consumption on the target memory system.

Memory system profiling is done via bandwidth–latency curves, for the baseline and target memory systems, along the lines outlined in Section 2.1. The precise method for obtaining these curves is given in Section 3, which describes the memory profiling microbenchmarks and their outputs.

CPU parameters are needed, alongside the application profiling (see below), to characterize the relationship between memory system latency and execution time. This relationship is dependent on the processor's ability to hide memory latency by overlapping memory accesses with independent instructions. As detailed in Section 4.3, the PROFET performance model therefore requires some basic parameters of the processor under study: re-order buffer (ROB) capacity, miss information status holding register (MSHR) capacity and minimum theoretical cycles-per-instruction (CPI).

Application profiling is done on the baseline memory system, and consists of executing the application and profiling it using hardware performance counters. *Application performance profiling* obtains the number of CPU cycles, number of instructions, number of last-level cache (LLC) misses and the read and write memory bandwidths. *Application power profiling* measures the total power consumption using integrated or external power measurement infrastructure, and memory-related power parameters using performance counters. Since the application's behavior changes over time, application profiling is done by sampling over regular time intervals, which we refer to as segments. Further details on application profiling are given in Section 4.1.

Memory power parameters characterize the baseline and target memory systems, in terms of the power consumption in various operational modes, idle state and power-down states, as well as the energy consumption for various operations such as read and write transfers, row buffer hits and misses. These figures are typically provided by the memory device manufacturers [28].

2.4 Performance, power and energy estimation

Figure 3 gives an overview of the whole process of performance, power and energy estimation. Since application profiling involves collecting a trace over the program's execution, the PROFET performance and power models are run for each segment (time interval) in the trace. This gives the predicted execution time, power and energy consumption of each segment. Summing over time gives the final execution time and energy for the whole application. The application's average power demand is total energy divided by total execution time.

The **PROFET performance model** reads the application performance information from the profiling trace-file and determines the application's position on the bandwidth–latency curve for the baseline memory system. As described in detail in Sections 4.2, 4.3 and 4.5, PROFET then estimates the application's position on the memory bandwidth–latency curve for the target memory system, and uses it to predict the application's performance on the target memory system.

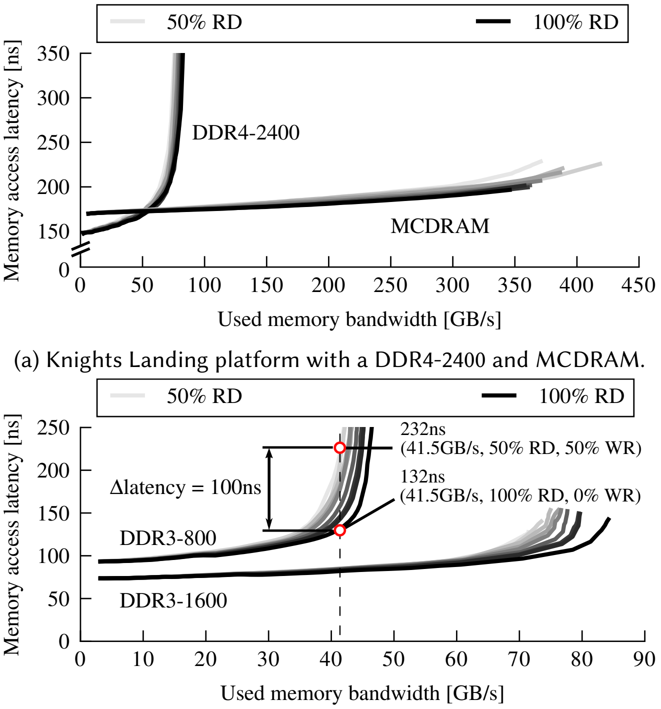
The **PROFET power model** estimates the power consumption of the target memory system using the application performance profiling and the memory power parameters. Finally, the **PROFET energy model** is done based on the output of the performance and power models. Due to a lack of space, the detailed description and evaluation of the PROFET power and energy models are presented in Appendices A.1, A.2 and B.1.

3 MEMORY SYSTEM PROFILING

The baseline and target memory systems are characterized using bandwidth–latency curves. For mature technologies, these curves are measured on a real platform. For emerging memory devices that are not yet available in off-the-shelf servers, the bandwidth–latency curve can be measured on a developer board with a prototype of the new device [2], or alternatively it can be provided by the manufacturer.

The bandwidth–latency curve is determined using a pointer-chasing microbenchmark designed to measure latency [34] running concurrently with a derivative of the STREAM benchmark [27] that was modified to vary the load on the memory system. Currently, profiling of a single memory system configuration, e.g., DDR3-1600, is performed in approximately 15 minutes (see Appendix B.2.3).

Although Section 2.1 plots a single bandwidth–latency curve, in reality a single memory system has a family of curves that depend on the ratio between read and writes in the overall memory traffic. As an example, Figure 4 shows the measured bandwidth–latency curves for the Knights Landing and Sandy Bridge platforms, as the proportion of reads is varied between 50% and 100%. The lightest curves correspond to 50% reads and the darkest curves correspond to 100% reads. Instead of the single bandwidth–latency curve per memory system that was illustrated in Figure 1, we now see a family of curves. When the used memory bandwidth is low or moderate, the read fraction has negligible impact on the memory access latency and the bandwidth–latency curves practically overlap. As the stress to the memory system increases, however, the read fraction starts



(a) Knights Landing platform with a DDR4-2400 and MCDRAM.

(b) Sandy Bridge platform with DDR3-800/1066/1333/1600. DDR3-1066 and DDR3-1333 are excluded to improve the visibility.

Fig. 4. Bandwidth–latency curves for the platforms under study. Memory access latency w.r.t. used memory bandwidth *cannot* be approximated with a single curve — as the used memory bandwidth increases, memory traffic read/write composition makes a significant latency impact.

to have a significant impact on latency. For example, in Figure 4b, at an aggregate bandwidth of 41.5 GB/s, the (read) latency with 100% reads is 132 ns, but the read latency with 50% reads and 50% writes is 232 ns, an increase of 100 ns (76%). In general, for all experiments we did, shown in Figure 4, curves with a higher percentage of writes (lighter curves) are located higher (at higher latency) on the chart. The main reason is that write requests incur additional delays that are not required by memory reads [19].[2] Increasing the proportion of write requests therefore reduces the sustainable bandwidth and increases the loaded latency.

Recently, Clapp et al. [4] also did a preliminary analysis of bandwidth–latency curves for different memory frequencies (DDR3-1333 and DDR3-1600) and different read-to-write ratios (3:1 and 2:1). Based on an analysis of four curves, the authors conclude that it is sufficient to use a single, generic memory bandwidth–latency curve for different frequencies and memory traffic compositions. Our analysis is based on numerous measurements on a wide range of DDR3, DDR4 and MCDRAM frequencies, with fine-grain changes in the read-to-write ratio. Our findings show that different memory frequencies have fundamentally different bandwidth–latency curves with different shapes and different lead-off and maximum memory access latencies. We also show that the read-to-write ratio may have a significant impact on memory access latency. Directly contrary to the conclusion

[2] *Write Recovery time* or t_{WR} is the minimum delay between the end of a write and the next precharge command. The *Write To Read delay time* or t_{WTR} is the minimum time interval between a memory write and a consecutive read.

of Clapp et al. [4], our study shows that the relationship between bandwidth and latency cannot be approximated with a single curve. To the best of our knowledge, ours is the first study of memory system read latency that makes this conclusion.

4 PERFORMANCE MODEL

This section presents the PROFET analytical model that predicts the application's performance. We start, in Section 4.1, by outlining the application characteristics that must be measured on the baseline system. Then, in Section 4.2, we introduce the problem with a simple case, that of an in-order processor. Next, in Section 4.3, we analyse a complex out-of-order processor. Section 4.4 completes the analysis of out-of-order processor performance as a function of latency. Finally, in Section 4.5, we explain how PROFET combines this latency–performance characterization with the bandwidth–latency curves to obtain the estimate, with error bars, of the application performance on the target memory system.

4.1 Application profiling

As outlined in Section 2.3, the application's execution is divided into segments at regular time intervals. For each segment, we measure, using performance measuring counters, the number of cycles, number of instructions, read last-level cache (LLC) misses, used memory bandwidth, and the overall fraction of reads. These parameters and the notation used in the paper are listed in Table 1.

Table 1. PROFET performance model input parameters

Input parameter	Symbol
Number of cycles	Cyc_{tot}
Number of Instructions	Ins_{tot}
Application read LLC misses	$Miss_{\text{LLC}}$
Used memory bandwidth for total traffic	$BW^{(1)}_{\text{used}}$
Fraction of reads in total traffic	$Ratio_{\text{R/W}}$

The used memory bandwidth, BW_{used}, and fraction of reads, $Ratio_{\text{R/W}}$, include all memory accesses, whether issued by the application or the prefetcher, since both types of accesses cause contention and have a similar impact on memory system read latency. In contrast, $Miss_{\text{LLC}}$ only includes LLC read misses issued by the application. This parameter is used to estimate how the memory read latency impacts application performance, and only application read misses have a direct performance impact.

In order to determine the duration of the sampling interval, we analyzed the tradeoff among the measurement overhead, trace-file size and PROFET accuracy. An interval of 1 s was selected because it provided high accuracy (see Section 7) while introducing negligible measurement overhead of below 1%. With this sampling interval, the trace-file size of the benchmarks used in the study is in the range of hundreds of megabytes, which is acceptable.

4.2 In-order processors

This section derives the relationship between latency and performance for a simple in-order CPU. In the interest of helping the reader to follow the formulas, we start by summarizing PROFET's inputs and outputs in Table 2.

Our analysis distinguishes between *Memory access latency*, *Memory access penalty* and *LLC miss penalty*. **Memory access latency**, Lat_{mem}, is the number of CPU cycles necessary for a single load

Table 2. Notation used in formulas: In-order processors

Description	Symbol
Inputs (in addition to Table 1)	
Memory access latency from bandwidth–latency curve	Lat_{mem}
Memory access penalty (Lat_{mem} minus LLC hit latency)	Pen_{mem}
Intermediate outputs	
Single LLC miss penalty (number of CPU stall cycles)	$Stalls_{\text{LLC}}$
Application cycles-per-instruction	CPI_{tot}
CPI component in the case of perfect LLC	CPI_0
CPI component due to LLC misses penalties	CPI_{LLC}
Outputs	
Application instructions-per-cycle ($1/CPI_{\text{tot}}$)	IPC_{tot}

instruction that reads data from the main memory. It is measured as part of the memory system profiling and given in the memory bandwidth–latency curve. **Memory access penalty**, Pen_{mem}, is the difference between the latency of a main memory access and the latency of an LLC hit. The values of Lat_{mem} and Pen_{mem} are inputs to PROFET. The values for the baseline memory system are found by looking up the application's used bandwidth, measured on the baseline memory system. The values for the target memory system are generated as explained in Section 4.5. Finally, **LLC miss penalty**, $Stalls_{\text{LLC}}$, is calculated by PROFET as the average number of cycles for which the CPU pipeline is stalled because of each LLC miss.

We start by partitioning the application cycles-per-instruction, CPI_{tot}, into two components [4, 8, 14, 22]: $CPI_{\text{tot}} = CPI_0 + CPI_{\text{LLC}}$. The first component, CPI_0, is the application's CPI for the hypothetical case of a 100% LLC hit rate. This component is not affected by the memory access latency. The second component, CPI_{LLC}, is due to execution stalls due to the LLC misses. Once we know the number of stall cycles to be attributed to each LLC miss, we can calculate its value as [3, 14]:

$$CPI_{\text{LLC}} = \frac{Miss_{\text{LLC}} \times Stalls_{\text{LLC}}}{Ins_{\text{tot}}} \tag{1}$$

We use the superscripts (1) and (2) to distinguish between the baseline and target memory systems, respectively:

$$\begin{aligned} &\text{Baseline memory:} \quad CPI_{\text{tot}}^{(1)} = CPI_0^{(1)} + CPI_{\text{LLC}}^{(1)} \\ &\text{Target memory:} \quad\;\; CPI_{\text{tot}}^{(2)} = CPI_0^{(2)} + CPI_{\text{LLC}}^{(2)} \end{aligned} \tag{2}$$

Since the memory access latency does not affect CPI_0, we have $CPI_0^{(1)} = CPI_0^{(2)}$. Therefore, $CPI_{\text{tot}}^{(2)}$ from Eq. 2 can be expressed as:

$$CPI_{\text{tot}}^{(2)} = CPI_{\text{tot}}^{(1)} + \left(CPI_{\text{LLC}}^{(2)} - CPI_{\text{LLC}}^{(1)}\right) \tag{3}$$

Next, we assume that a change in the memory system, which for this section is a change in the read access latency, does not change the number of instructions, Ins_{tot}, or the application's memory access pattern. This is a reasonable assumption for applications or computational kernels that do not use busy-waiting or dynamic scheduling. The assumption is valid for both in-order and out-of-order processors. A change in the memory access latency may affect the timeliness of the prefetcher, but it should not consistently affect its coverage or accuracy; in any case, we found this effect to be small.[3] We therefore also assume that the LLC miss rate is unaffected by the change in memory latency. In summary, we conclude that we do not need superscripts (1) or (2) on Ins_{tot} and $Miss_{\text{LLC}}$. We can therefore substitute Eq. 1 into Eq. 3 to obtain:

$$CPI^{(2)}_{\text{tot}} = CPI^{(1)}_{\text{tot}} + \frac{Miss_{\text{LLC}}}{Ins_{\text{tot}}} \times \left(Stalls^{(2)}_{\text{LLC}} - Stalls^{(1)}_{\text{LLC}}\right) \tag{4}$$

We have not yet assumed an in-order processor, so all the above equations are also true for out-of-order processors. For an in-order processor, we make the single observation that LLC misses directly lead to pipeline stalls, i.e.: $Stalls_{\text{LLC}} = Pen_{\text{mem}}$. Substituting this into Eq. 4 gives:

$$CPI^{(2)}_{\text{tot}} = CPI^{(1)}_{\text{tot}} + \frac{Miss_{\text{LLC}}}{Ins_{\text{tot}}} \times \left(Pen^{(2)}_{\text{mem}} - Pen^{(1)}_{\text{mem}}\right) \quad \text{for an in-order processor} \tag{5}$$

Eq. 5 is important because it shows that the difference in application CPI for different memory systems, $CPI^{(2)}_{\text{tot}}$ and $CPI^{(1)}_{\text{tot}}$, can be calculated based on the corresponding memory access penalties, $Pen^{(2)}_{\text{mem}}$ and $Pen^{(1)}_{\text{mem}}$. Finally, by replacing $IPC = 1/CPI$, the application performance on the target memory system is calculated as:

$$IPC^{(2)}_{\text{tot}} = \frac{1}{\frac{1}{IPC^{(1)}_{\text{tot}}} + \frac{Miss_{\text{LLC}}}{Ins_{\text{tot}}} \times (Pen^{(2)}_{\text{mem}} - Pen^{(1)}_{\text{mem}})} \quad \text{for an in-order processor} \tag{6}$$

4.3 Out-of-order processors

The analysis for out-of-order (OOO) processors is more complex because following an LLC miss the processor can continue executing independent instructions without immediately being stalled. In consequence, the number of stalls per LLC miss is no longer equal to the full memory access penalty, and it is typically strictly lower than it: $Stalls_{\text{LLC}} < Pen_{\text{mem}}$. In order to handle this inequality, it is necessary to introduce the additional symbols given in Table 3.

Table 3. Notation used in formulas: Out-of-order processors

Description	Symbol
Inputs	
Instructions in reorder buffer	Ins_{ROB}
Size of miss information status holding register (MSHR)	$MSHR$
Minimum CPI, equal to reciprocal of maximum IPC	CPI_{min}
Intermediate outputs	
Number of execution cycles overlapped with LLC miss stalls, due to OOO mechanism	Cyc_{ooo}
Number of instructions executed during LLC miss stalls, due to OOO mechanism	Ins_{ooo}
Cyc_{tot} component in the case of perfect LLC	Cyc_0
Memory level parallelism: Number of concurrent LLC misses (memory accesses)	MLP

[3] We measured the number of prefetches per instruction on our Sandy Bridge evaluation platform (Section 6.1). The overall difference between DDR3-800 and DDR3-1600 memory configurations across all benchmarks is less than 5%.

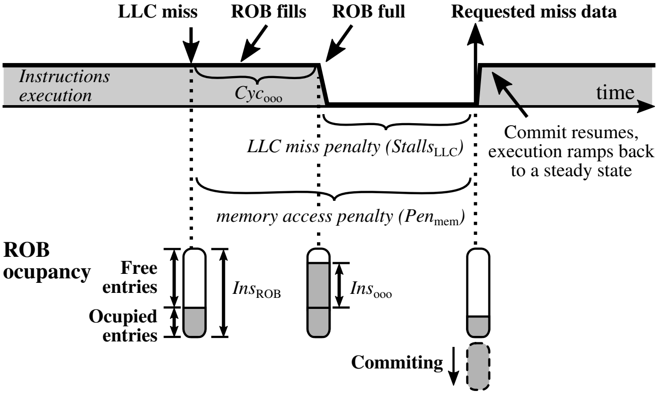


Fig. 5. In OOO processors, LLC misses overlap with the execution of the instructions independent of the missing data. The overlap depends on the number of independent instructions in the instruction window and number of free entries in ROB [22].

4.3.1 ***Isolated LLC miss.*** We first consider an isolated LLC miss. As illustrated in Figure 5, when an LLC miss occurs (isolated or not), the corresponding instruction must wait for data from memory, but the CPU pipeline continues issuing and executing independent instructions. Execution may halt, however, before the LLC miss is resolved, for two reasons. First, instruction issue may stop because the instruction window has filled with instructions, all of which are dependent, directly or indirectly, on the instruction waiting for data from main memory. Second, instruction commit may stop because the reorder buffer (ROB) has filled with instructions that cannot be committed until after the waiting instruction has itself been committed. The upper part of Figure 5 shows a timeline indicating whether instruction execution has halted, while the lower part of the figure shows snapshots of the ROB occupancy before an isolated LLC miss, while the processor is waiting for data, and after the data has been received. Following the LLC miss, the ROB is occupied with a certain number of instructions. The ROB begins to fill, as the processor executes independent instructions. At some point, either there are no more independent instructions or the ROB becomes full. In either case, instruction execution will stall. Once the LLC miss has been resolved and the LLC miss data are available, the instructions waiting for the data can be executed and committed. This allows the instructions in the ROB to be committed, so the processor can resume issuing and executing new instructions.

In Figure 5, the period after the LLC miss in which the processor is executing new independent instructions is labeled as Cyc_{ooo}.So, the number of stall cycles is given by [14, 22]:

$$Stalls_{LLC} = Pen_{mem} - Cyc_{ooo} \quad \text{for an isolated miss} \tag{7}$$

If execution is immediately halted following the LLC miss, then Cyc_{ooo} would equal zero, and the LLC miss penalty would equal the memory access penalty, as for the in-order case in Section 4.2.

The number of independent instructions that are executed during this period, of Cyc_{ooo}, is referred to as Ins_{ooo}, and indicated in the lower half of Figure 5. The connection between Cyc_{ooo} and Ins_{ooo} requires knowledge of the CPI over the period. Our analysis partitions the application execution in sampling segments of 1 second (detailed in Section 4.1), and considers there to be a steady average execution rate. So, during these execution segments the CPI equals its average rate of CPI_0, as detailed in Table 2 and the corresponding text. Therefore, Cyc_{ooo} can be calculated as:

$$Cyc_{ooo} = CPI_0 \times Ins_{ooo} \quad \text{for an isolated miss} \tag{8}$$

The factor of Ins_{ooo} depends on the number of independent instructions and the number of free instruction slots in the ROB. It is analyzed in detail in the next section.

4.3.2 *Estimating Ins_{ooo}.*

State-of-the-art architectures do not incorporate counters that can be used to measure the value of Ins_{ooo}. We therefore calculate bounds on its value and incorporate these bounds into the PROFET error estimate. We use the platform-specific parameters and the application's measured CPI.

The Ins_{ooo} lower bound is trivial: $Ins_{\mathrm{ooo}} \geq 0$, since OOO execution may stop immediately after the LLC miss and continue being stalled until the requested miss data arrives.

The Ins_{ooo} upper bound is calculated as the lower of two constraints. The first is the reorder buffer size, Ins_{ROB}, which corresponds to the maximum number of instructions that can be stored in the ROB [22]:

$$Ins_{\mathrm{ooo}}^{\mathrm{max1}} = Ins_{\mathrm{ROB}} \tag{9}$$

The ROB size is a characteristic on the target architecture. In our study, we analyze two architectures, as described in Section 6. In Sandy Bridge EP-2670 CPU the ROB comprises 168 entries, while in Intel Knights Landing Xeon Phi 7230 it has 72 entries.

The second upper bound is determined by the maximum number of instructions that can be executed during the LLC miss. In this scenario, the whole Pen_{mem} is covered by the OOO execution, so Cyc_{ooo} would equal Pen_{mem}. Therefore, since Ins_{ooo} is calculated as Cyc_{ooo}/CPI_0 (Eg. 8), we can combine the two equations to find that the maximum number of instructions that the processor will execute in this time is Pen_{mem}/CPI_0. Since the second upper bound assumes that OOO execution covers the whole memory access penalty, there cannot be any stalls due to LLC misses, i.e., CPI_{LLC} would equal 0. Therefore, since the application's overall CPI is defined to be $CPI_{\mathrm{tot}} = CPI_0 + CPI_{\mathrm{LLC}}$, it must be (in this case) that CPI_{tot} equals CPI_0. Combining these facts, the final form of the second upper-bound, given in terms of inputs to PROFET, becomes:

$$Ins_{\mathrm{ooo}}^{\mathrm{max2}} = Pen_{\mathrm{mem}} \times \frac{Ins_{\mathrm{tot}}}{Cyc_{\mathrm{tot}}} \tag{10}$$

The overall upper bound on Ins_{ooo} is the minimum of the two limits:

$$Ins_{\mathrm{ooo}}^{max} = \min\left(Ins_{\mathrm{ROB}}, \quad Pen_{\mathrm{mem}} \times \frac{Ins_{\mathrm{tot}}}{Cyc_{\mathrm{tot}}}\right) \tag{11}$$

Since the value of Ins_{ooo} can be anywhere between its bounds we consider Ins_{ooo} to be a free parameter and perform a sensitivity analysis when calculating other dependent parameters.

4.3.3 *Overlapping LLC misses: Impact of memory level parallelism.*

Previously, in Section 4.3.1, specifically in Eq. 7, we considered the case of an isolated read LLC miss. This section now considers the general case, in which after an LLC miss occurs, and while the corresponding instruction is waiting for data from memory, the CPU pipeline generates one or more additional LLC misses. This situation is illustrated in Figure 6. Any stall cycles that occur should be counted once per group of overlapping LLC misses rather than once per LLC miss, which was the case for Eq. 7. The number of concurrent LLC misses is typically known as the memory level parallelism, and is denoted MLP [3, 13]. The penalty per LLC miss is therefore given by the number of stall cycles divided by MLP [3, 22]:[4]

[4] LLC misses can also overlap with front-end miss events such as instruction cache miss, branch misprediction, etc. These overlaps, however, tend to be rare leading to an insignificant performance impact [9].

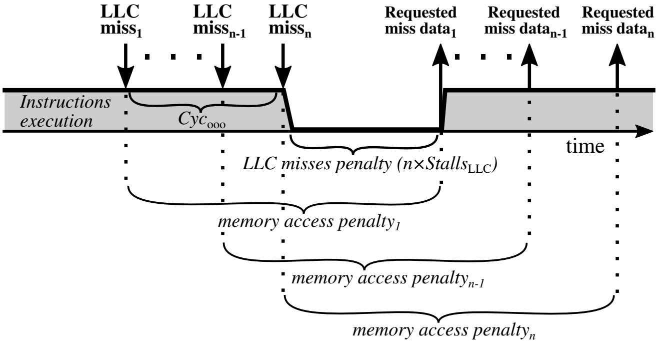

LLC miss$_1$
LLC miss$_{n-1}$
LLC miss$_n$
Requested miss data$_1$
Requested miss data$_{n-1}$
Requested miss data$_n$
Instructions execution
Cyc_{ooo}
time
LLC misses penalty (n×$Stalls_{LLC}$)
memory access penalty$_1$
memory access penalty$_{n-1}$
memory access penalty$_n$

Fig. 6. Handling overlapping LLC misses in an OOO processor: the penalty of a single miss is divided by a number of concurrent LLC misses [22].

$$\begin{aligned} Stalls_{\text{LLC}} &= \frac{1}{MLP} \times (Pen_{\text{mem}} - Cyc_{\text{ooo}}) \\ &= \frac{1}{MLP} \times (Pen_{\text{mem}} - CPI_0 \times Ins_{\text{ooo}}) \end{aligned} \tag{12}$$

Karkhanis et al. [22] analyze this in detail and show that Eq. 12 is correct independently of the moment in which second, third, or subsequent LLC misses occur, as long as they occur within the Cyc_{ooo} interval. If MLP equals 1, then the above equation becomes identical to Eq. 7; so Eq. 12 covers both cases, that of isolated and overlapping LLC misses.

Current processors cannot directly measure MLP, so it must be estimated based on the parameters that are available. We derive lower and upper bounds on MLP and a point estimate.

The *MLP* lower and upper bounds can be computed starting from the equation $CPI_{\text{tot}}^{(1)} = CPI_0 + CPI_{\text{LLC}}^{(1)}$, then substituting $CPI_{\text{LLC}}^{(1)}$ from Eq. 1 and $Stalls_{\text{LLC}}^{(1)}$ from Eq. 12:

$$CPI_{\text{tot}}^{(1)} = CPI_0 + \frac{Miss_{\text{LLC}} \times \left(Pen_{\text{mem}}^{(1)} - CPI_0 \times Ins_{\text{ooo}}\right)}{Ins_{\text{tot}} \times MLP} \tag{13}$$

Rearranging to isolate MLP and writing as a function to make clear which values are unknown gives:

$$MLP(Ins_{\text{ooo}}, CPI_0) = \frac{\frac{Miss_{\text{LLC}}}{Ins_{\text{tot}}} \times (Pen_{\text{mem}}^{(1)} - CPI_0 \times Ins_{\text{ooo}})}{CPI_{\text{tot}}^{(1)} - CPI_0} \tag{14}$$

This equation expresses MLP, which we want to know, in terms of Ins_{ooo}, the free variable that we will vary later, and CPI_0, which is unknown but can be bounded. The lower bound on CPI_0 is CPI_{min}, the reciprocal of the processor's *highest* theoretical IPC. The upper bound on CPI_0 is $CPI_{\text{tot}}^{(1)}$, since CPI_0 was defined to be one (of two) components contributing to $CPI_{\text{tot}}^{(1)}$. Now that CPI_0 is bounded, and assuming a value of Ins_{ooo}, it is possible to use Eq. 14 to obtain the range of potential values of MLP, either via a sweep on CPI_0 between its lower and upper bounds or using differential calculus.

A second upper bound on MLP is the size of the Miss Information Status Holding register (MSHR) [25]. The MSHR is the hardware structure that keeps information about in-flight cache misses, so they can be resolved once the corresponding data arrives. Its size is CPU-specific; e.g. it is 10 for Sandy Bridge [17] and 12 for KNL [21].

The *MLP* point estimate is derived by assuming that the application's behavior is uniform (in a sense to be clarified below) over the sampling segment. Specifically, we assume that the number of LLC misses per instruction is homogeneous across the time segment, in which case it must equal $Miss_{\mathrm{LLC}}/Ins_{\mathrm{tot}}$. In the period between the LLC miss and the arrival of its data, the processor executes Ins_{ooo} instructions, so with a constant rate of LLC misses, the total number of *additional* LLC misses is $\frac{Miss_{\mathrm{LLC}}}{Ins_{\mathrm{tot}}} \times Ins_{\mathrm{ooo}}$. The value of Ins_{ooo} is a free parameter, as described in Section 4.3.2, so the value being calculated here is a function of that parameter. In order to account for the first LLC miss, which has not yet been counted, the point estimate for the total number of LLC misses, as a function of Ins_{ooo}, to which the stall cycles must be attributed, is:

$$\widehat{MLP}(Ins_{\mathrm{ooo}}) = \frac{Miss_{\mathrm{LLC}}}{Ins_{\mathrm{tot}}} \times Ins_{\mathrm{ooo}} + 1 \tag{15}$$

Note that $\widehat{MLP}(Ins_{\mathrm{ooo}})$ is a point estimate for MLP based on the available information. If the point estimate is outside the valid range, between the lower and upper bounds described above, then it is corrected to lie in the range.

4.4 Performance as a function of latency

This section completes the analysis of out-of-order processor performance as a function of latency. We start by repeating Eq. 4, which gives the predicted CPI in terms of $Stalls_{\mathrm{LLC}}$:

$$CPI_{\mathrm{tot}}^{(2)} = CPI_{\mathrm{tot}}^{(1)} + \frac{Miss_{\mathrm{LLC}}}{Ins_{\mathrm{tot}}} \times \left(Stalls_{\mathrm{LLC}}^{(2)} - Stalls_{\mathrm{LLC}}^{(1)}\right) \tag*{(4 again)}$$

As remarked at the beginning of Section 4.3, in comparison with an in-order processor, an out-of-order processor has a more complex expression for $Stalls_{\mathrm{LLC}}$, and this was given in Eq. 12:

$$Stalls_{\mathrm{LLC}} = \frac{1}{MLP} \times (Pen_{\mathrm{mem}} - CPI_0 \times Ins_{\mathrm{ooo}}) \tag*{(12 again)}$$

Finally we replace the MLP parameter in this equation with the point estimate in Eq. 15:

$$\widehat{MLP}(Ins_{\mathrm{ooo}}) = \frac{Miss_{\mathrm{LLC}}}{Ins_{\mathrm{tot}}} \times Ins_{\mathrm{ooo}} + 1 \tag*{(15 again)}$$

In fact, as explained in Section 4.3.3, this value is restricted to lie between the lower and upper bounds given in that section. For the sake of clarity, we consider the more common case for which it is not necessary.

Combining Eq. 4, Eq.12 and Eq.15, and assuming that Ins_{tot}, $Miss_{\mathrm{LLC}}$, CPI_0, Ins_{ooo} and MLP do not change when moving from one memory system configuration to another, then $CPI_{\mathrm{tot}}^{(2)}$ can be calculated as:

$$CPI_{\mathrm{tot}}^{(2)} = CPI_{\mathrm{tot}}^{(1)} + \frac{Pen_{\mathrm{mem}}^{(2)} - Pen_{\mathrm{mem}}^{(1)}}{Ins_{\mathrm{ooo}} + Ins_{\mathrm{tot}}/Miss_{\mathrm{LLC}}} \tag{16}$$

This equation is written in terms of the memory access penalty, Pen_{mem}, but at the system level, outside a detailed analysis of a particular processor's pipeline, only Lat_{mem} is relevant. Recall that Pen_{mem} was defined to be the memory access latency, Lat_{mem} minus the cost of an LLC hit. We note, therefore, that the expression $Pen_{\mathrm{mem}}^{(2)} - Pen_{\mathrm{mem}}^{(1)}$ is equal to $Lat_{\mathrm{mem}}^{(2)} - Lat_{\mathrm{mem}}^{(1)}$. Taking account of this and rewriting in terms of the IPC instead of the CPI gives:

$$IPC_{\mathrm{tot}}^{(2)} = \frac{IPC_{\mathrm{tot}}^{(1)}}{1 + IPC_{\mathrm{tot}}^{(1)} \times \frac{Lat_{\mathrm{mem}}^{(2)} - Lat_{\mathrm{mem}}^{(1)}}{Ins_{\mathrm{ooo}} + Ins_{\mathrm{tot}}/Miss_{\mathrm{LLC}}}} \tag{17}$$

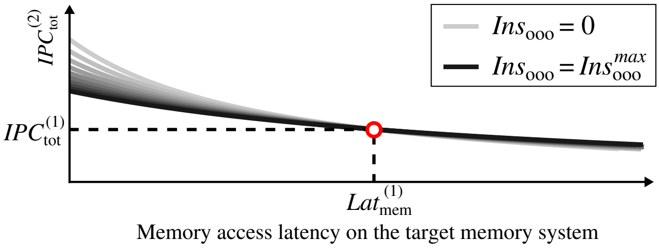


Fig. 7. Performance as a function of the memory access latency. The different curves arise by varying the unknown parameter Ins_{ooo} within its bounds. Please read the text before interpreting this figure.

The various values in Eq. 17, $IPC_{\text{tot}}^{(1)}$, Ins_{tot}, and $Miss_{\text{LLC}}$ are known because they were measured on the baseline memory configuration. All other inputs to PROFET, such as Ins_{ROB}, $MSHR$, $CPI_{\min}$ (see Table 3) appear in the upper and lower bounds of Ins_{ooo}.[5]

Eq. 17 is plotted in Figure 7. The x axis is the target system memory latency, $Lat_{\text{mem}}^{(2)}$, and the y axis is the predicted IPC, $IPC_{\text{tot}}^{(2)}$. Eq. 17 is a function of the independent parameter Ins_{ooo}, which we cannot measure or calculate exactly. We bounded its value in the previous section, and varying it between the lower and upper bounds gives the family of curves shown in the figure. Note that the case of $Ins_{\text{ooo}} = 0$ corresponds to an in-order processor. This can be seen by comparing Eq. 17 and Eq. 6. As indicated on the figure, when the target memory latency is the same as the baseline memory latency, $Lat_{\text{mem}}^{(1)}$, PROFET correctly "predicts" the measured IPC to be that of the baseline system, $IPC_{\text{tot}}^{(1)}$.

It is easy to be misled by Figure 7. For instance, a decrease in the memory latency by a fixed value, e.g. reducing the lead-off load penalty by 10 ns, is **not** equivalent to simply moving by 10 ns to the left on the x axis. This is because, as seen in the figure, such a change will result in an increase in the IPC, which will itself cause an *increase in the used memory bandwidth*, for reasons explained in the next section. This increase in used bandwidth will cause a movement to the right in the memory's bandwidth–latency curve and therefore *increase the memory system latency*, counterbalancing the original decrease in memory system latency. We address this problem in the next section.

4.5 Performance estimation — the ultimate step

This section completes the PROFET performance model. We start from the bandwidth–latency curves described in Section 3, which give the loaded memory access latency as a function of used memory bandwidth. We then combine these curves with the analysis in Section 4.4, which gives application performance as a function of the loaded memory latency. Doing so, in the right way, gives a prediction of the performance on the target memory system, with error bars.

The solution will be explained through Figure 8. The x axis is the used memory bandwidth and the y axis is the memory access latency. We show the measured bandwidth–latency curves for the Knights Landing platform, exactly as in Figure 4a. As before, the lightest curves correspond to 50% reads and 50% writes, and the darkest curves correspond to 100% reads. Now, however, since we are analysing a specific application segment, the proportion of reads is known (it is $Ratio_{\text{R/W}}$), so we know which curve from the family to select. The selected curve, which corresponds to $Ratio_{\text{R/W}}$, is shown as a dashed white curve.

We now turn to Figure 7, and use the latency–performance plot to construct a latency–*bandwidth* plot, i.e. to find the used memory *bandwidth* as a function of the memory access latency. This

[5] Some of the input parameters appear only in the upper or lower bounds of MLP, which are not in Eq. 17 but considered in the full PROFET model.

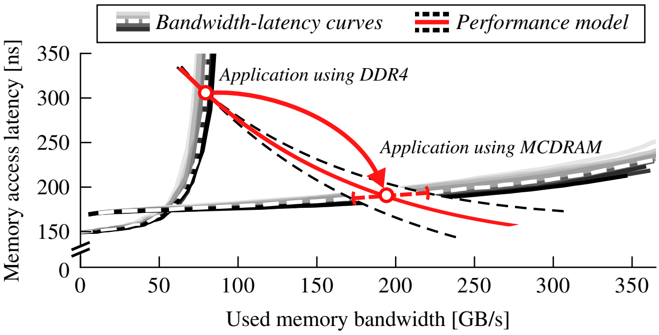


Fig. 8. Graphical interpretation of a performance estimation as a merged solution of Sections 3 and 4.4.

is because the total number of memory accesses performed over the application's execution is a constant as argued in Section 4.2. The total number of accesses is evaluated, for the baseline memory system, by multiplying bandwidth by time, giving $BW^{(1)}_{\text{used}} \times (Cyc^{(1)}_{\text{tot}}/Freq_{\text{CPU}})$. Dividing this expression by the execution time on the target memory system gives:

$$BW^{(2)}_{\text{used}} = \frac{BW^{(1)}_{\text{used}} \times (Cyc^{(1)}_{\text{tot}}/Freq_{\text{CPU}})}{(Cyc^{(2)}_{\text{tot}}/Freq_{\text{CPU}})} = \frac{BW^{(1)}_{\text{used}}}{IPC^{(1)}_{\text{tot}}} \times IPC^{(2)}_{\text{tot}} \tag{18}$$

We therefore obtain a plot of bandwidth vs. latency simply by multiplying the value on the y axis of Figure 7 by the factor $BW^{(1)}_{\text{used}}/IPC^{(1)}_{\text{tot}}$. The axes now match those in Figure 8 except they are transposed. We therefore transpose the bandwidth vs. latency plot, by swapping the x and y axes, and superimpose it onto Figure 8. This gives the family of lines for the PROFET performance model.

When the application runs on a memory system, it must be located on the memory system's bandwidth–latency curve *and* on one of the PROFET performance model curves that was just added. It must therefore be located on the intersection of these curves, as indicated in Figure 8. For the baseline memory system, we find that all PROFET performance model curves intersect the bandwidth–latency curve in the same place, at the bandwidth measured on the real system.

Each pair of bandwidth–latency and PROFET performance model curves will intersect in exactly one place. There cannot be more than one intersection because the memory system's bandwidth–latency curve is increasing (as a function of latency) whereas the application's performance model curve is decreasing (as a function of latency). In addition, there must be an intersection point, since the application's curve decreases from a very high latency necessary to get a small used memory bandwidth whereas the memory's bandwidth–latency curve increases to a very high latency close to the maximum sustainable bandwidth.

In summary, we start from the target memory system's bandwidth–latency curve and Eq. 17, which defines a family of PROFET performance model curves. We perform a sweep of the valid range for Ins_{ooo}, and for each value, find the intersection of its performance model curve with the target bandwidth–latency curve. To find this intersection we use the bisection method. This point gives a bandwidth on the y axis, which can be converted to an IPC by rearranging Eq. 18. Varying Ins_{ooo} in this way gives the minimum, maximum and point estimate for IPC, from which the number of cycles and execution time can also be easily deduced. Recall that the discussion so far is related to a single segment (time interval) of the application. Summing over all segments gives the predicted minimum, maximum and point estimate execution time for the whole application. Execution of the complete PROFET prediction for the whole application is very fast. For example, for the benchmarks under study the performance estimate for each target memory system is completed within seconds.

4.6 Novelties of the presented analytical model

Since our PROFET analytical model is based on CPI stack analysis as widely used for performance modeling [14], it is important to emphasize the contributions of our work beyond the previous studies.

Computation of the CPI component that corresponds to the execution stalls due to the LLC misses (Eq. 1) is described by previous studies [3, 4, 14]. Hennessy and Patterson [14] and Karkhanis and Smith [22] also analyze execution of independent instructions Ins_{ooo} after the LLC miss (Eq. 7) and define some of the Ins_{ooo} bounds (Eq. 9). Finally, the MLP and its impact on CPI stack analysis (Eq. 12) are also well explored by the community [3, 4, 6, 10, 22].

CPI stack, Ins_{ooo} and MLP are the foundation of various analytical models that quantify the performance impact of the main memory latency [3, 6, 10, 22]. The previous analytical models, however, have one great challenge: they require detailed application profiling, which can be performed only with hardware simulators. The main objective of our work is to avoid the use of the simulators, and to develop an analytical model based *only* on the parameters that can be obtained or derived from performance counters measurements on actual platforms. This requires a novel approach to the MLP estimate, presented in Eq. 13–15. In addition to this, we also present additional Ins_{ooo} bounds in Eq. 10 and Eq. 11. Finally, to the best of our knowledge, this is the first study that combines analysis of application performance as a function of memory access latency (Sec 4.4) with bandwidth–latency curves (Sec 4.5) and that shows that its analysis leads to a unique solution.

5 POWER AND ENERGY ESTIMATION

Similarly to performance estimation, we develop the PROFET power and energy models. Based on the application profiling and memory power parameters, these models predict the variation of the system power and energy consumption due to the change of the memory systems. PROFET's power and energy models are both validated on the Sandy Bridge E5-2670 server running SPEC2006 and scientific HPC applications.[6] As for the PROFET performance model, the error of the power and energy estimation is low: less than 2% for power and less than 3% for energy consumption. Due to the lack of space, the detailed description of the PROFET power and energy models is moved to Appendix A, while their evaluation is given in Appendix B.1.

6 METHODOLOGY

In this section, we present the hardware platforms and benchmarks used in the evaluation of PROFET. We also list the tools used for the application profiling and server power measurements. Finally, we summarize the main steps of the PROFET evaluation process.

6.1 Hardware platforms

We evaluate PROFET on Sandy Bridge-EP E5-2670 and Knights Landing (KNL) Xeon Phi platforms. The most important features of the platforms are summarized in Table 4.

The Sandy Bridge-EP server is a representative of mainstream high-performance computing (HPC) servers, and it is still in use, especially in smaller Tier-0 systems [1]. In the server under study, we were able to test four memory frequencies: DDR3-800, DDR3-1066, DDR3-1333 and DDR3-1600, and we used these configurations to evaluate PROFET's performance, power and energy models.

The Intel Knights Landing (KNL) Xeon Phi platform [36] is an emerging platform that combines two types of memory with different memory bandwidths and access latencies: DDR4 DIMMs and 3D-stacked MCDRAM [36]. Since it uses two types of memories, the system offers three modes of

[6] The PROFET power and energy models were not developed for the KNL server, because we lacked reliable MCDRAM power parameters. This research is a part of ongoing work.

Table 4. The most important features of experimental platforms

Platforms	**Sandy Bridge** **E5-2670**	**Knights Landing** **Xeon Phi 7230**
Sockets	2	1
Cores per socket	8	64
CPU freq. [GHz]	3.0	1.3
L1i, L1d	32 kB, 32 kB	32 kB, 32 kB
L2	512 kB	1 MB
L3	20 MB	/
Memory conf.	4 chann.	8 chann. MCDRAM
per socket	DDR3-800/1066/1333/1600	6 chann. DDR4-2400
Memory capacity	64 GB	16 GB MCDRAM 96 GB DDR4

operation: *cache* mode, *flat* mode and *hybrid* mode. In our experiments, we use *flat* mode, in which the DDR4 and MCDRAM are configured as separate NUMA nodes, and we execute our workloads either in DDR4 or MCDRAM memory.

6.2 Benchmarks

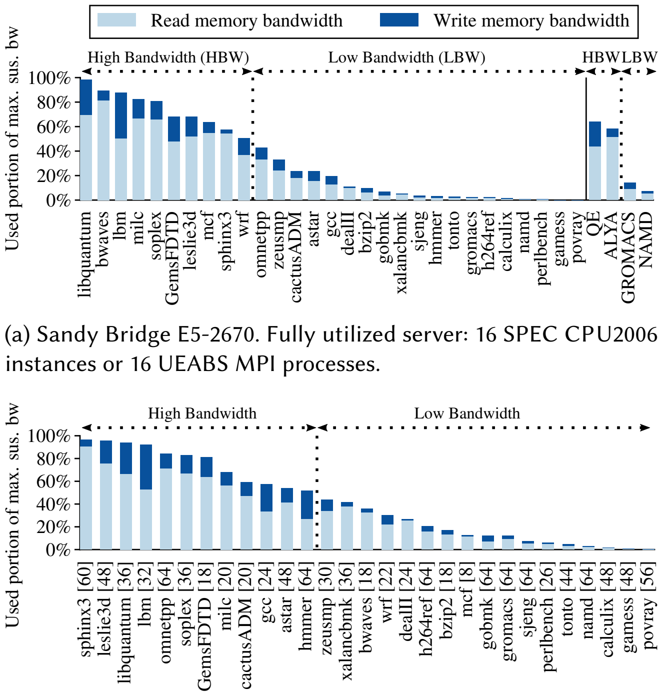
We evaluated PROFET on a set of SPEC CPU2006 benchmarks [37] and scientific HPC applications. The scientific applications are selected from the Unified European Application Benchmark Suite (UEABS) [30]. These applications are parallelized using Message Passing Interface (MPI) and are representative of production applications running on HPC systems in Europe. We choose four applications: ALYA, representative of the computational mechanics codes, and GROMACS, NAMD, and Quantum Espresso (QE) computational chemistry applications. The remaining UEABS applications could not be executed because their input dataset sizes exceed the main memory capacity of our hardware platforms.

In all the experiments on Sandy Bridge platform, we fully utilize the available 16 CPU cores (2 sockets, 8 cores each): we execute 16 copies of each SPEC CPU2006 benchmark, or 16 application processes for each UEABS applications. The KNL platform comprises 16 GB of the MCDRAM, which was insufficient to execute any of the UEABS application. Also, for each SPEC CPU2006 benchmark, we had to determine the maximum number of the instances whose cumulative memory footprint would fit into the MCDRAM. In the charts, this number of benchmark instances is specified in parentheses after the benchmark name.

In order to quantify the level of stress that our workloads put on the memory system, we measure their memory bandwidth on the platforms under study. Figure 9 shows the bandwidth utilization, relative to the maximum sustained memory bandwidth measured with STREAM benchmark [27]. When reporting the evaluation of PROFET in Section 7, we emphasize the results for the high-bandwidth benchmarks with over 50% of the used memory bandwidth. These benchmarks are the most affected by the changes in the memory system, and therefore the most challenging to model.

6.3 Tools and methodology

Application profiling requires measurements of the CPU cycles, instructions, LLC misses, read and write memory bandwidths, as well as the row-buffer access statistics, number of page activations and page misses, and number of cycles spent in memory power-down states. All these inputs are



(a) Sandy Bridge E5-2670. Fully utilized server: 16 SPEC CPU2006 instances or 16 UEABS MPI processes.

(b) Knights Landing Xeon Phi. The MCDRAM capacity limits the number of the benchmarks instances, specified in the square brackets.

Fig. 9. The workloads under study show a wide range of memory bandwidth utilization, and different ratios of the Read and Write memory traffic.

measured by the hardware counters and the LIKWID performance tool suite [39]. The counters used in the study are widely available in mainstream HPC servers [16, 18].

We used a Yokogawa WT230 [5] power meter to measure the server power consumption. The power meter measures the voltage and current at the power plug to calculate the power consumption of the whole server, including power supply, motherboard with all its components, CPUs, and memory. The measurements were sampled on one second time period. The energy consumption was calculated by summing the power consumption over the execution time.

7 EVALUATION

Evaluation of PROFET is done in four steps. First, we execute a benchmark on the *baseline memory system*, e.g., Sandy Bridge server with DDR3-800. In this run, we measure the benchmark performance, power and energy consumption, and collect all the hardware counters needed for prediction using PROFET. Second, we use PROFET to *estimate* the benchmark performance, power and energy on the *target memory configuration*, e.g., DDR3-1600. Third, we change the platform memory configuration from the baseline to the target memory, e.g., from DDR3-800 to DDR3-1600. This requires changing the BIOS settings for the Sandy Bridge, and changing the execution NUMA node for the Knights Landing platform. Finally, we *execute* the benchmark on the *target memory*

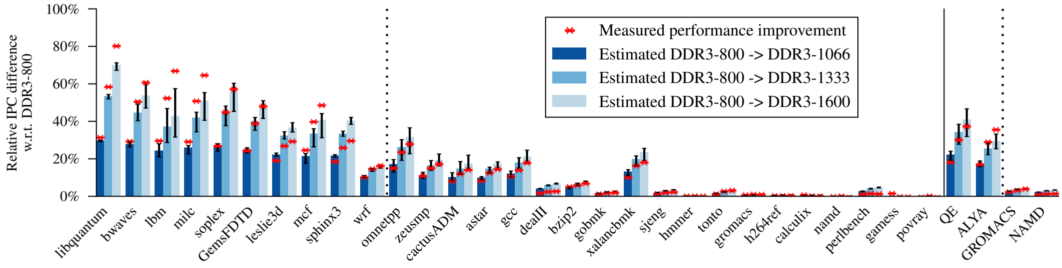


Fig. 10. Sandy Bridge, DDR3-800 → 1066/1333/1600: Changing the DRAM frequency has a significant performance impact. PROFET's performance model estimations are precise, with low error bars, and accurate, with small difference from the values measured on the actual hardware.

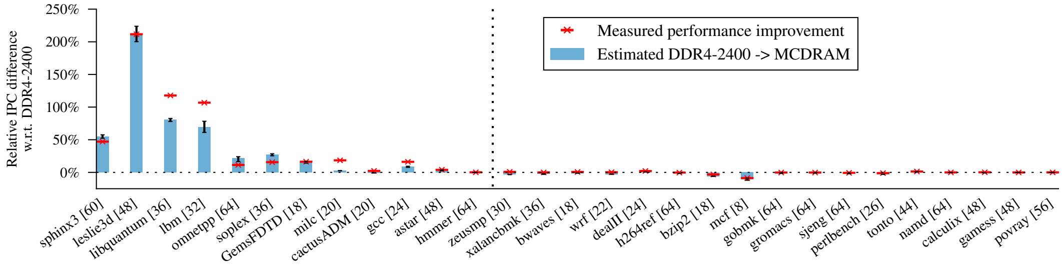


Fig. 11. Knights Landing, DDR4-2400 → MCDRAM: Despite the wide range of the performance variation, between −9% (*mcf*) and 212% (*leslie3d*), the model shows high accuracy. Smaller KNL reorder buffer leads to a smaller range of the Ins_{ooo} sensitivity analysis, and therefore more precise performance prediction, i.e., smaller error bars.

system, measure the *actual* performance, power and energy consumption, and compare them with the values estimated by PROFET.

7.1 Sandy Bridge: DDR3-800 → 1066/1333/1600

The results for the Sandy Bridge server are displayed in Figure 10. For each benchmark we plot three sets of bars, one per DRAM frequency change: DDR3-800→1066/1333/1600. As described in Section 4.3, since we could not determine the exact value of the Ins_{ooo} parameter in the experimental platform, we performed a sensitivity analysis by varying Ins_{ooo} between 0 and Ins_{ooo}^{max}, and doing the performance estimate for each possible value. The solid bars correspond to the mean performance estimate, while the error bars show the lowest and highest estimated performance. In addition, we plot the actual performance improvement measured on the real platform, marked with a (red) cross marker for each experiment.

First, we can see that the error bars, i.e., ranges of the estimated performance are narrow. Across the high-bandwidth benchmarks, the average width of the error bars is only 2.6%, 5.5% and 7.4%, for DDR3-1066/1333/1600 respectively. Across the low-bandwidth benchmarks, the average width of the error bars is even lower, at 1.1%, 1.6% and 2%. This means that, although for the architectures under study we cannot determine the precise value of the Ins_{ooo} parameter, we can, in all cases, apply a sensitivity analysis and obtain a narrow range of estimated performance. As expected, the high-bandwidth benchmarks, which are more sensitive to the CPU's ability to hide memory latencies through memory parallelism, have a greater sensitivity to Ins_{ooo}, leading to wider, but still acceptable, error bars. Second, PROFET's predictions are highly accurate. The average difference from the performance measured on the actual hardware for DDR3-1066/1333/1600 frequencies is 1.8%, 3.8% and 5.1% for the high-bandwidth benchmarks, and it drops to just 1%, 1.3% and 1.6% over the low-bandwidth benchmarks. Finally, the presented results show that the DRAM frequency increase indeed has a significant performance impact. For example, increasing the DRAM frequency from the baseline DDR3-800 to the target DDR3-1600 causes average performance improvement of 22%, and it reaches 80% for the *libquantum* benchmark. Therefore it is important to understand the relation between the available memory bandwidth and the overall application performance, which is the main objective of our work.

7.2 Knights Landing: DDR4-2400 → MCDRAM

Figure 11 shows the estimated and measured performance improvement of the Knights Landing Xeon Phi with high-bandwidth MCDRAM with respect to the DDR4-2400 memory. Again, PROFET shows high precision. Actually, the width of the estimation error bars is only 3.2% for high-bandwidth benchmarks and 0.4% for low-bandwidth benchmarks, both of which are significantly smaller than the corresponding figures for the Sandy Bridge system. The main reason for this is the 72-instruction reorder buffer in the KNL platform, w.r.t. the 168 instructions in the Sandy Bridge. A smaller reorder buffer leads to a narrower range for the Ins_{ooo} sensitivity analysis, and therefore a more precise performance prediction. In the case of low-bandwidth benchmarks, additional reason for small error bars is that their overall performance sensitivity on the change from the baseline to the target memory system is low. Therefore, PROFET also predicts low performance difference with close values for minimum and maximum of the Ins_{ooo} sensitivity analysis. In their case, Ins_{ooo} parameter from Eq. 17 has negligible impact on the estimation result, either because of the close values of $Lat_{\text{mem}}^{(2)}$ and $Lat_{\text{mem}}^{(1)}$ or because of the low number of LLC misses in Eq. 17.

The MCDRAM provides 4.2-fold higher bandwidth over DDR4, which leads to significant performance improvement for the high-bandwidth benchmarks, up to 212% improvement for the *leslie3d*. However, the MCDRAM also has a 23 ns higher lead-off latency (see Figure 4a), that penalizes benchmarks with low and moderate memory bandwidth requirements. Actually, for *bzip* and *mcf*, execution on the MCDRAM leads to performance loss. For the *mcf* benchmark, this performance loss reaches a non-negligible 9%.

Despite the wide range of the performance variation, between −9% and 212%, when moving from DDR4 to the MCDRAM, PROFET shows high accuracy. The difference between PROFET's performance estimates and the measurements on the actual hardware is 7% for high-bandwidth benchmarks, and drops down to 1.6% for the low-bandwidth benchmarks. Also, PROFET's predictions accurately distinguish between the benchmarks that significantly benefit from the MCDRAM, and the benchmarks that show negligible performance improvements or even performance loss. This confirms that PROFET properly considers both segments of the memory bandwidth–latency curves: the constant latency segment, close to the lead-off memory latency, and the exponential segment, close to the memory bandwidth saturation point.

7.3 Additional evaluations

Section 7.1 summarizes the evaluation of the **performance model** on the Sandy Bridge when increasing the DRAM frequency from DDR3-800 to DDR3-1066/1333/1600. Appendix B.1 extends these results with the evaluation of the **power and energy model** on the same platform and DRAM configurations. PROFET's power and energy estimates are evaluated versus measurements of an external power meter taken on the actual Sandy Bridge server. As remarked above, PROFET's power and energy models were not developed for the KNL server, because we lacked the reliable

MCDRAM power parameters. As a part of the ongoing work, we are exploring different ways to estimate these parameters.

The evaluation results show that PROFET's predictions are **highly-accurate**, typically with only 2% difference from the performance, power and energy consumption measurements on the actual hardware. PROFET provides accurate estimations even when the baseline and target memory systems have **fundamentally different bandwidth–latency curves**, with different lead-off latency and with an n-fold difference in the available bandwidth.

In Appendix B.2, we compare the performance estimates of PROFET with ZSim+DRAMSim2 hardware simulators. The comparison is done based on the actual measurements on the Sandy Bridge platform. The presented results show that PROFET has much better accuracy than the simulators. Also, PROFET's estimations closely follow the trend of the actual measurements, while the simulated results show completely different trends. Appendix B.2 also summarizes additional advantages of PROFET. PROFET is faster than the hardware simulators by *three orders of magnitude*, so it can be used to analyze production HPC applications, arbitrarily sized systems, and numerous design options, within a practical length of time. Also, PROFET does not require detailed CPU modeling since the CPU functionality, including actual data prefetcher and out-of-order engine, is already accounted for by the application profiling. Finally, PROFET can be used on various platforms as long as they support the required application profiling. PROFET was initially developed for the Sandy Bridge platform, and later we evaluated it for the KNL server. Adjustment of PROFET to the KNL system was trivial, requiring changes to only a few hardware parameters.

8 RELATED WORK

CPU and memory analytical models: Numerous studies propose analytical models as an alternative to cycle-accurate hardware simulation. We summarize the ones that are directly related to our work.

Karkhanis and Smith [22] present an OOO processor model that estimates the performance impact of instruction window size, branch misprediction, instruction cache misses and data cache misses. The model is validated versus detailed superscalar OOO CPU simulation and the authors conclude that, although the model provides much less data than detailed simulation, it still provides insights on what is going on inside the processor. Published in 2004, this work became the foundation for numerous advanced processor modeling approaches. Eyerman et al. [10] extend the work of Karkhanis and Smith [22] and develop the *mechanistic model and interval analysis*. The interval analysis breaks the total execution time into intervals based on the miss events, branch mispredictions and TLB/cache misses, and then predicts the execution time of each interval. Genbrugge et al. [12] propose *interval simulation* which accelerates multi-core simulation via a combination of the high-level mechanistic analytical model [10] and detailed simulation. The mechanic analytical model is used to estimate the core-level performance between two miss events, while the miss events are determined through the simulation of branch predictor and the entire memory hierarchy: private per-core caches and TLBs, shared caches, cache coherence, network on chip, memory controller, and main memory. Finally, Van den Steen et al. [6] present various enhancements of the interval model [10]. First, the study incorporates architecture-independent application profiling, so the application can be profiled once and then simulated on any given platform. The authors also demonstrate that the analytical model can be connected to the McPAT power tool [26] for estimation of the processor power and energy consumption. Finally, the authors make first steps in memory system modeling by considering congestion on the memory bus. Unlike our study, however, they do not take into account contention in the memory controller and memory device itself, which are more challenging.

The greatest challenge of the presented modeling approaches is that, although they do not require simulation to carry out the performance prediction, they do require detailed application profiling, which can be performed only with hardware simulators. Even with recent advances in hardware performance counters, we are still far away from being able to read values such as, for example, the number of cold, capacity and conflict cache miss. Since these methods require simulation results, even for application profiling, a significant effort is required to set up and tune for a target architecture, a serious amount of simulation time, and potentially high simulation errors. Our approach avoids these issues by using only those parameters that can be read using performance counters on real hardware. This is the greatest advantage of our work compared with previously-mentioned studies. Limiting the application profiling to the performance counters available on real hardware requires a novel approach to infer various application parameters, such as the MLP or number of executed OOO instructions. Therefore, although we start with the same foundation as the previous studies, as detailed in Section 4.6, estimation of the important application parameters and the overall performance calculation are fundamentally different from previously-presented models.

The second important difference of our work compared with the previous studies is the treatment of the main memory access latency. The previous studies use the memory access latency obtained from a detailed simulation of the whole memory hierarchy [12] or simply use a constant latency [6, 10, 22]. Our study performs a detailed analysis of memory bandwidth–latency curves and uses these curves as an intrinsic part of the overall performance estimation, as detailed in Section 4.5.

Third, unlike previous studies, PROFET encompasses data prefetching. This is an important difference because prefetching may have significant impact on the application performance, behavior and memory bandwidth usage.

Fourth, we complement the PROFET performance model with power and energy consumption estimates, and evaluate all three PROFET models against fully-utilized actual HPC servers with multi-threaded or multi-programmed benchmark execution. The previous models are validated versus the same simulators used for the application profiling. This means that the previous evaluations overlook potentially high errors of application profiling on a simulation versus the actual hardware. Finally, most of the previous studies [6, 10, 22] build and validate the models for single-core processors.

Workload characterization and memory DVFS: The performance impact of memory bandwidth and latency is frequently estimated by workload characterization studies and memory DVFS proposals. Memory models that could quantify this impact are missing, or not publicly available, so some studies develop their own models [4, 7]. These models are developed for very specific tasks in the context of the larger studies and they successfully fulfil their objectives. Still it is questionable whether they can be applied generally because of two main limitations. First, the modeled CPU and memory systems are much simpler than state-of-the-art production platforms. Second, the models are not validated versus any real hardware or hardware simulators, so it is difficult to quantify the errors they may introduce. Even so, we consider these studies as very valuable for any follow-up on this topic because they analyze different approaches for main memory modeling and share their experiences.

9 CONCLUSIONS

This study presents PROFET, an analytical model that quantifies the impact of the main memory on application performance and system power and energy consumption. PROFET is based on memory system profiling and instrumentation of an application execution on a real platform with a baseline memory system. By running on the real platform, PROFET handles many aspects (e.g. prefetcher and out-of-order engine) that are oversimplified in state-of-the-art methods. The outputs from PROFET are the predicted performance, power and energy consumption on the target memory.

PROFET is evaluated on two actual platforms: Sandy Bridge-EP E5-2670 and Knights Landing Xeon Phi platforms with various memory configurations. The evaluation results show that PROFET's predictions are very accurate — the average difference from the performance, power and energy measured on the actual hardware is typically only about 2%. We also compare PROFET's performance predictions with simulation results for the Sandy Bridge-EP E5-2670 system with ZSim and DRAMSim2 simulators. PROFET shows significantly better accuracy while being three orders of magnitude faster than the hardware simulators. We release PROFET's source code and all input data required for memory system and application profiling [31]. The released model is ready to be used on high-end Intel platforms, and we would encourage the community to use it, adapt it to other platforms, and share their own evaluations.

ACKNOWLEDGMENTS

This work was supported by the Spanish Ministry of Science and Technology (project TIN2015-65316-P), Generalitat de Catalunya (contracts 2014-SGR-1051 and 2014-SGR-1272), Severo Ochoa Programme (SEV-2015-0493) of the Spanish Government; and the European Union's Horizon 2020 research and innovation programme under ExaNoDe project (grant agreement No 671578) and EuroEXA project (grant agreement No 754337); the U.S. Department of Defense under Contract FA8075-14-D-0002-0007, TAT 15-1158; and the U.S. National Science Foundation under Award 1642424.

A POWER AND ENERGY MODELING

In this appendix, we give the detailed description of the PROFET power and energy models.

A.1 Power modeling

Apart from performance, power and energy demand are important system constraints. In modern HPC systems, the memory subsystem contributes 10–16% of the total server power consumption [11]. It is therefore valuable to quantify the trade-offs in power and energy consumption due to the change of the memory system.

As mentioned in Section 5, we analyse the difference in total system power consumption when we move from baseline to target memory system. In our study, we assume that when the memory system changes, the biggest impact on total system power consumption is the change of the main memory power consumption. Hence, we focus on modeling the power consumption of the memory subsystem and consider that power consumption of the rest of the system does not change [7]. Estimating power and energy consumption requires measurements of the total platform power consumption, ratio of time spent in memory power-down states (active standby, precharge power-down and self-refresh states in our system) and row-buffer access statistics (rate of page hits and page misses). Table 5 summarizes the symbols used in the formulas for power and energy estimation.

To calculate the components of memory power consumption, we use Micron's guide for calculating power consumption of DDR3 memory systems [28]. Apart from the parameters in Table 5, Micron's power consumption guide requires *IDD* currents, memory system voltage and DIMM timing parameters, which are detailed in DIMMs documentation [29].[7] The PROFET power model was not developed for the KNL server, because we lacked of the reliable MCDRAM power parameters. The estimation of these power parameters is part of ongoing work.

Micron's power consumption guide defines how to calculate power consumption of individual read or write memory accesses and DRAM power-down states. In our power analysis, we begin

[7] During the evaluation on different memory frequencies (DDR3-800/1066/1333/1600) we used the same DIMMs. Each memory frequency, however, uses different timing and *IDD* current parameters.

Table 5. Notation used in formulas: Power modeling

Description	Symbol
Input parameters	
Total platform power consumption	P_{tot}
Percentage of time spent in active standby state	t_{act}
Percentage of time spent in precharge power-down state	t_{ppd}
Percentage of time spent in self-refresh state	t_{sr}
Percentage of row-buffer hits	p_{hit}
Percentage of row-buffer misses	p_{miss}
Used memory bandwidth for read/write traffic	$BW_{\text{used}}^{rd/wr,(1)}$
Intermediate outputs	
Total memory power	P_{mem}
Power consumption of the rest of the system, apart from the memory	P_{rest}
Operational memory power	P_{op}
Background memory power	P_{bg}
Memory power in active standby state	P_{act}
Memory power in precharge power-down state	P_{ppd}
Memory power in self-refresh state	P_{sr}
Total memory read/write operations power	$P_{\text{rd/wr}}$
Power of memory refresh operations	P_{ref}
Duration of sampling segment	T_{sample}
Number of read/write memory accesses on a sampling segment	$N_{\text{access}}^{rd/wr}$
Energy on termination resistors for a single read/write memory access	$E_{\text{term}}^{rd/wr}$
Single memory read/write access energy	$E_{\text{access}}^{rd/wr}$
Energy of a read/write row buffer miss access	$E_{\text{miss}}^{rd/wr}$
Energy of a read/write row buffer hit access	$E_{\text{hit}}^{rd/wr}$

from total platform power consumption and expand its components down to these basic DRAM operations. We start by dividing the total platform power consumption into two components, power of the memory system and power of the rest of the platform apart from the memory [7]: $P_{\text{tot}} = P_{\text{mem}} + P_{\text{rest}}$. As in the analysis of the performance in Section 4, we use the superscripts (1) and (2) to distinguish between the P_{tot} and its components in the baseline and target memory systems, respectively:

$$\begin{aligned} &\text{Baseline memory:} \quad P_{\text{tot}}^{(1)} = P_{\text{mem}}^{(1)} + P_{\text{rest}}^{(1)} \\ &\text{Target memory:} \quad P_{\text{tot}}^{(2)} = P_{\text{mem}}^{(2)} + P_{\text{rest}}^{(2)} \end{aligned} \tag{19}$$

Since we assume that the power of the rest of the system stays the same, we can write $P_{\text{rest}}^{(2)} = P_{\text{rest}}^{(1)}$.[8] Therefore, total platform power consumption in the target memory system can be expressed as:

$$P_{\text{tot}}^{(2)} = P_{\text{tot}}^{(1)} + (P_{\text{mem}}^{(2)} - P_{\text{mem}}^{(1)}) \tag{20}$$

[8] Quantifying the impact of a change from baseline to target memory system on P_{rest} is a part of ongoing work.

In further analysis we focus on the P_{mem}. It comprises two components, background power P_{bg} and operational power P_{op}:

$$P_{\text{mem}} = P_{\text{bg}} + P_{\text{op}} \tag{21}$$

Background power accounts for the current state of the memory system. There are several possible states of the memory system, depending on which power-down states are used. On our experimental system, there are three supported memory power states: *active standby*, *precharge power-down* and *self-refresh*. They differ in terms of power consumption and the transition latency to the active state. In *active standby state*, the memory device consumes the highest power P_{act} but executes the commands immediately, without any latency penalty. *Precharge power-down* state consumes power P_{ppd}, which is less than P_{act}, with a moderate latency penalty. In *self-refresh* mode, memory consumes the least power P_{sr}, but has a significant latency penalty for coming back to active standby mode. Multiplying each of these powers with the corresponding time share spent in each of them (t_{act}, t_{ppd} and t_{sr}, respectively),[9] and summing these products gives the background power P_{bg}:

$$P_{\text{bg}} = t_{\text{act}} \times P_{\text{act}} + t_{\text{ppd}} \times P_{\text{ppd}} + t_{\text{sr}} \times P_{\text{sr}} \tag{22}$$

Operational power presents the sum of power consumptions while reading or writing the data, plus the power consumption of the refresh operations:

$$P_{\text{op}} = P_{\text{rd}} + P_{\text{wr}} + P_{\text{ref}} \tag{23}$$

Components P_{rd} and P_{wr} present power consumptions of all read or write memory accesses, respectively, on a sampling interval. Expanding these components further leads to the power consumptions of individual read or write memory accesses, which can be calculated using the Micron's power consumption guide. However, using the power consumption of individual read or write memory accesses to calculate P_{rd} and P_{wr} is not trivial. These individual reads or writes are interleaved and overlapped in time. In order to sum the powers of individual reads or writes, we have to know their distribution on intervals which are the orders of magnitude of 1 ns, which is infeasible in current hardware platforms.

To mitigate this problem we calculate the energy of individual read or write memory access. Using energy instead of power implies that we do not have to know the distribution of memory accesses on a sampling segment to calculate the sum of energies of all the individual reads or writes. This way, we calculate P_{rd} and P_{wr} in two steps. First, we sum the energies of all the individual reads or writes on the sampling segment. Second, we divide this cumulative energy from the previous step with the duration of the sampling segment. If an energy of a single read or write memory access is $E_{\text{access}}^{rd/wr}$ and there are $N_{\text{access}}^{rd/wr}$ number of reads or writes on a sampling segment T_{sample}, we can write:

$$P_{\text{rd}} = \frac{\sum_{i=1}^{N_{\text{access}}^{rd}} E_{\text{access,i}}^{rd}}{T_{\text{sample}}} = \frac{E_{\text{access}}^{rd} \times N_{\text{access}}^{rd}}{T_{\text{sample}}} \tag{24}$$

$$P_{\text{wr}} = \frac{\sum_{i=1}^{N_{\text{access}}^{wr}} E_{\text{access,i}}^{wr}}{T_{\text{sample}}} = \frac{E_{\text{access}}^{wr} \times N_{\text{access}}^{wr}}{T_{\text{sample}}} \tag{25}$$

Number of reads or writes on the sampling segment can be measured with memory bandwidth hardware counters. A single read or write memory access transfers the amount of data defined as width of the memory bus (64 bits) multiplied by number of bursts (8), so 8 Bytes × 8 bursts =

[9] Please note that $t_{\text{act}} + t_{\text{ppd}} + t_{\text{sr}} = 1$.

64 Bytes of data. Therefore, number of reads or writes equals the total read or write traffic during the sampling segment, divided by the size of a single memory access:

$$N^{rd}_{\text{access}} = \frac{BW^{rd}_{\text{used}} \times T_{\text{sample}}}{64\text{ B}} \tag{26}$$

$$N^{wr}_{\text{access}} = \frac{BW^{wr}_{\text{used}} \times T_{\text{sample}}}{64\text{ B}} \tag{27}$$

Using Eq. 26 and 27 with Eq. 24 and 25, we get:

$$P_{\text{rd}} = E^{rd}_{\text{access}} \times \frac{BW^{rd}_{\text{used}}}{64\text{ B}} \tag{28}$$

$$P_{\text{wr}} = E^{wr}_{\text{access}} \times \frac{BW^{wr}_{\text{used}}}{64\text{ B}} \tag{29}$$

Energy per single memory access accounts for read or write operation with its sub-operations. It also includes the energy on termination resistors, which are common in DDR devices. Our Sandy Bridge experimental platform uses *adaptive open-page policy* [15, 19], therefore performed sub-operations depend whether the target row in memory array was open or closed when accessing it. If the target row was open, it is a *row-buffer hit* access and it consumes only the energy for reading or writing the data. When the target row is closed, there are two scenarios. The first scenario is that there is no other opened row in the same bank and initially the energy accounts for opening a row and reading or writing the data. This row will be eventually closed after a time-out, so precharge energy should be added afterwards. The second scenario is that if there is an opened row (which is not the target one) in the same bank, it has to be closed first. It accounts the energy for precharging and closing the opened row, activating the target row and reading or writing the data. Both scenarios include same sub-operations from the energy point of view and we consider them as *row-buffer miss* case in further analysis. Since we measure the ratio of row-buffer hits p_{hit} and row-buffer misses p_{miss}[10] w.r.t. total number of accesses, the energy per memory access can be represented as:

$$E^{rd}_{\text{acc}} = E^{rd}_{\text{hit}} \times p_{\text{hit}} + E^{rd}_{\text{miss}} \times p_{\text{miss}} + E^{rd}_{\text{term}} \tag{30}$$

$$E^{wr}_{\text{acc}} = E^{wr}_{\text{hit}} \times p_{\text{hit}} + E^{wr}_{\text{miss}} \times p_{\text{miss}} + E^{wr}_{\text{term}} \tag{31}$$

The energy parameter E_{miss} represents the row-buffer miss energy, including opening the row, sending or receiving the data and precharging the row. E_{hit} represents the row-buffer hit energy and it includes sending or receiving the data. E_{term} represents the energy on termination resistors. These parameters are calculated using Micron's power consumption guide.

Now that we have all the power components, we can include all of them from Equations 21 to 31 into Eq. 32 and calculate $P^{(2)}_{\text{tot}}$ on the target memory system. Before this step, we have to make two assumptions. First assumption is that the ratio of time spent in memory power-down states stays the same on baseline and target memory system. Hence, $t^{(2)}_{\text{act}} = t^{(1)}_{\text{act}}$, $t^{(2)}_{\text{ppd}} = t^{(1)}_{\text{ppd}}$ and $t^{(2)}_{\text{sr}} = t^{(1)}_{\text{sr}}$. Second assumption is that row-buffer access statistics does not change from baseline to target memory system. So, $p^{(2)}_{\text{hit}} = p^{(1)}_{\text{hit}}$ and $p^{(2)}_{\text{miss}} = p^{(1)}_{\text{miss}}$. These assumptions are reasonable, since we assume that Ins_{tot}, $Miss_{\text{LLC}}$, CPI_0, $Ratio_{\text{R/W}}$ and memory access pattern do not change

[10] Note that $p_{\text{hit}} + p_{\text{miss}} = 1$.

from baseline to target memory system (detailed in Section 4.2). So, the final solution for $P_{\text{tot}}^{(2)}$ on the target memory system is given in the Eq. 32:

$$
\begin{aligned}
P_{\text{tot}}^{(2)} &= P_{\text{tot}}^{(1)} + (P_{\text{mem}}^{(2)} - P_{\text{mem}}^{(1)}) = P_{\text{tot}}^{(1)} + \left(P_{\text{bg}}^{(2)} + P_{\text{op}}^{(2)} - (P_{\text{bg}}^{(1)} + P_{\text{op}}^{(1)})\right) \\
&= P_{\text{tot}}^{(1)} + P_{\text{bg}}^{(2)} - P_{\text{bg}}^{(1)} + P_{\text{op}}^{(2)} - P_{\text{op}}^{(1)} \\
&= P_{\text{tot}}^{(1)} + P_{\text{bg}}^{(2)} - P_{\text{bg}}^{(1)} + P_{\text{ref}}^{(2)} - P_{\text{ref}}^{(1)} + P_{\text{rd}}^{(2)} - P_{\text{rd}}^{(1)} + P_{\text{wr}}^{(2)} - P_{\text{wr}}^{(1)} \\
&= P_{\text{tot}}^{(1)} + \overbrace{t_{\text{act}} \times (P_{\text{act}}^{(2)} - P_{\text{act}}^{(1)}) + t_{\text{ppd}} \times (P_{\text{ppd}}^{(2)} - P_{\text{ppd}}^{(1)}) + t_{\text{sr}} \times (P_{\text{sr}}^{(2)} - P_{\text{sr}}^{(1)})}^{P_{\text{bg}}^{(2)} - P_{\text{bg}}^{(1)} \quad \text{(Eq. 22)}} \\
&\quad + (P_{\text{ref}}^{(2)} - P_{\text{ref}}^{(1)}) + \overbrace{\frac{1}{64\text{B}} \times \left(BW_{\text{used}}^{rd,(2)} \times (E_{\text{hit}}^{rd,(2)} \times p_{\text{hit}} + E_{\text{miss}}^{rd,(2)} \times p_{\text{miss}} + E_{\text{term}}^{rd,(2)})\right)}^{P_{\text{rd}}^{(2)} \quad \text{(Eq. 28 and Eq. 30)}} \\
&\quad - \underbrace{\frac{1}{64\text{B}} \times \left(BW_{\text{used}}^{rd,(1)} \times (E_{\text{hit}}^{rd,(1)} \times p_{\text{hit}} + E_{\text{miss}}^{rd,(1)} \times p_{\text{miss}} + E_{\text{term}}^{rd,(1)})\right)}_{P_{\text{rd}}^{(1)} \quad \text{(Eq. 28 and Eq. 30)}} \\
&\quad + \overbrace{\frac{1}{64\text{B}} \times \left(BW_{\text{used}}^{wr,(2)} \times (E_{\text{hit}}^{wr,(2)} \times p_{\text{hit}} + E_{\text{miss}}^{wr,(2)} \times p_{\text{miss}} + E_{\text{term}}^{wr,(2)})\right)}^{P_{\text{wr}}^{(2)} \quad \text{(Eq. 29 and Eq. 31)}} \\
&\quad - \underbrace{\frac{1}{64\text{B}} \times \left(BW_{\text{used}}^{wr,(1)} \times (E_{\text{hit}}^{wr,(1)} \times p_{\text{hit}} + E_{\text{miss}}^{wr,(1)} \times p_{\text{miss}} + E_{\text{term}}^{wr,(1)})\right)}_{P_{\text{wr}}^{(1)} \quad \text{(Eq. 29 and Eq. 31)}}
\end{aligned}
\tag{32}
$$

A.2 Energy modeling

Once we have the performance and the power consumption estimations, we can estimate the total system energy consumption with the target memory system. In general, energy is defined as the integral of power over time:

$$
E_{\text{tot}} = \int_{0}^{t_{\text{tot}}} P_{\text{tot}}(t)\text{d}t
$$

In our experiments, we measure and analyse power consumption on sampling segments of 1 s. Hence, we represent total energy from our experiments in a discrete form:

$$
E_{\text{tot}} = \sum_{i=1}^{N} P_{\text{tot,i}} \times \Delta t_{\text{i}} \tag{33}
$$

The parameter Δt_{i} is the duration of the sampling segment, and equals $\Delta t_{\text{i}}{}^{(1)} = 1$ s on a baseline memory system (detailed in Section 4.1). During $\Delta t_{\text{i}}{}^{(1)}$ segment, $Ins_{\text{tot,i}}^{(1)}$ instructions are executed. As we mentioned in Section 4.2, we assume that Ins_{tot} does not change from baseline to target memory system, therefore $Ins_{\text{tot,i}}^{(1)} = Ins_{\text{tot,i}}^{(2)}$. However, duration of the corresponding time interval $\Delta t_{\text{i}}{}^{(2)}$ on the target memory system is not the same as $\Delta t_{\text{i}}{}^{(1)}$. This implies that $\Delta t_{\text{i}}{}^{(2)}$ on the target memory system is inversely proportional to the estimated performance improvement:

$$\Delta t^{(2)} = \frac{IPC^{(1)}_{\mathrm{tot,i}}}{IPC^{(2)}_{\mathrm{tot,i}}} \times \Delta t_{\mathrm{i}}{}^{(1)} \tag{34}$$

Finally, total energy on the target memory system is:

$$E^{(2)}_{\mathrm{tot}} = \sum_{i=1}^{N} P^{(2)}_{\mathrm{tot,i}} \times \Delta t^{(2)}_{\mathrm{i}} = \sum_{i=1}^{N} P^{(2)}_{\mathrm{tot,i}} \times \frac{IPC^{(1)}_{\mathrm{tot,i}}}{IPC^{(2)}_{\mathrm{tot,i}}} \times \Delta t_{\mathrm{i}}{}^{(1)} \tag{35}$$

Parameter $P^{(2)}_{\mathrm{tot,i}}$ can be calculated using the Equation 32, and $IPC^{(2)}_{\mathrm{tot,i}}$ can be calculated using the PROFET performance model from Section 4.

B ADDITIONAL EVALUATIONS

Due to the lack of space, Sections 7.1 and 7.2 of the paper show a subset of the PROFET evaluation experiments performed in the study. Rest of the evaluation results are presented in this appendix. Appendix B.1 shows the evaluation of the PROFET power and energy model for Sandy Bridge when increasing the DRAM frequency from DDR3-800 to DDR3-1066/1333/1600. PROFET's power and energy estimates are evaluated versus measurements of an external power meter taken on the actual Sandy Bridge server. These results are a direct extension of the Sandy Bridge performance model evaluation presented in Section 7.1. Afterwards, in Appendix B.2 we compare PROFET's performance estimates with the outcomes of the ZSim+DRAMSim2 hardware simulator.

B.1 Sandy Bridge: DDR3-800 → 1066/1333/1600 Power and energy estimation

In Section 7, we showed the performance evaluation of PROFET on a Sandy Bridge server when increasing the DRAM frequency from DDR3-800 to DDR3-1066/1333/1600. In this section, we present the evaluation results for system-level estimation of power and energy consumption.

B.1.1 System power. Figure 12 shows the estimated and measured system power. As in Figure 10, for each benchmark we plot three sets of bars, one per DRAM configuration: DDR3-800 → 1066/1333/1600. Also, as in all previous evaluation charts, we plot PROFET's point estimate (solid bars) and estimation bounds (error bars), and the power consumption measured on the actual server (cross markers).

Although some of the high-bandwidth benchmarks, *libquantum* to *GemsFDTD* show a moderate prediction error of 3–4%, in general the power prediction error of PROFET is small, below 1% on average. The most important finding of the results presented in Figure 12 is that the significant increment in the DRAM frequency causes very small change in the overall server power consumption. For example, increasing the DRAM frequency from DDR3-800 to 1600 (100% increment) causes average power increment of only 2%. Even if we focus on the high-stress memory benchmarks, the power increment is still below 5%. This is not surprising. As assumed in PROFET's power model, when changing the DRAM frequency, the most important impact on total system power consumption is the change of the main memory power consumption. Although the relative change of the memory power itself could be significant, this is still a small portion of the overall server power [11].

B.1.2 Energy consumption. Estimated and measured changes in the system energy consumption when increasing the DRAM frequency from DDR3-800 to DDR3-1066/1333/1600 are given in Figure 13. The results show that the DRAM frequency has a significant impact on the overall energy consumption. For example, increasing the DRAM frequency from DDR3-800 to DDR3-1600 leads to

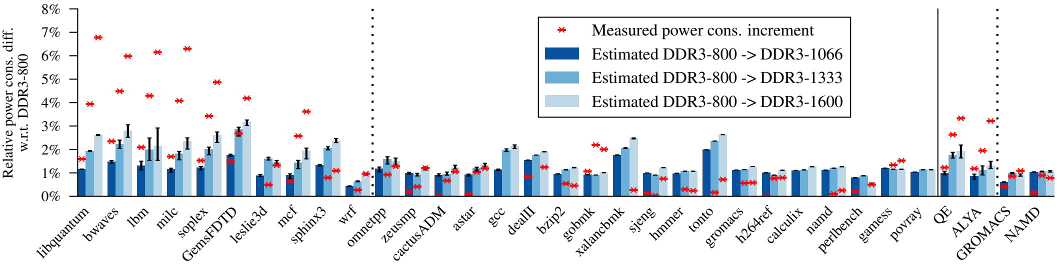


Fig. 12. Sandy Bridge, DDR3-800 → 1066/1333/1600: DRAM frequency has a minor impact on the overall server power consumption. Increasing DRAM frequency from DDR3-800 to DDR3-1600 (100% increment) causes average power increment of only 2%.

average energy savings of 13% and reaches 41% savings for the *libquantum* benchmark. The results also show that PROFET's energy predictions are precise, with narrow error bars, and accurate, with average prediction error of below 2%.

The presented results and findings are not surprising. Our previous results showed that increasing the DRAM frequency causes significant execution time reductions, and only few percent server power increment. Since, energy is the integral of the power consumption over the application execution time, it is reasonable to expect significant energy savings. Also, it is anticipated that the energy predictions are precise and accurate, since they are derived from the performance and power estimations.

B.2 PROFET vs. Hardware simulator

Novel memory systems are typically explored using hardware simulators. In this section we compare PROFET's performance estimates with ZSim+DRAMSim2 hardware simulators.

B.2.1 Experimental. We compared the hardware simulators and PROFET on the **Intel Sandy Bridge** platform described in Section 6.1. Although the Sandy Bridge is a main-stream HPC architecture released eight years ago (January 2011), finding a CPU simulator that accurately models this architecture was not trivial. After extensive search and analysis of the available options, we decided to use the **ZSim simulator**. ZSim [33] was initially developed to simulate Intel Westmere architecture (released in 2008), and it was recently upgraded simulate the Sandy Bridge [40]. Another reason for selecting ZSim was its simulation speed. The ZSim was designed for simulation of large-scale

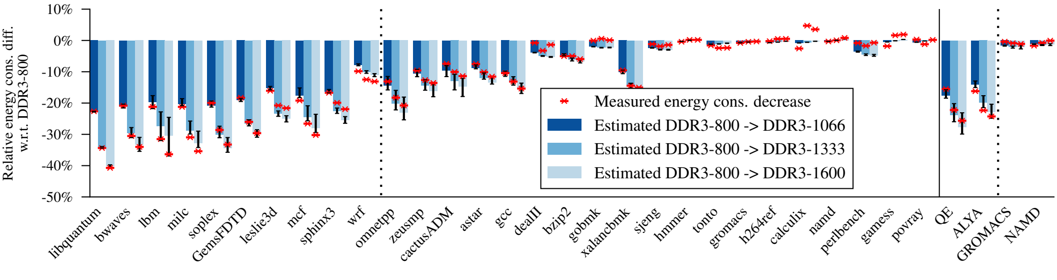


Fig. 13. Sandy Bridge, DDR3-800 → 1066/1333/1600: DRAM frequency has a significant impact on the system energy consumption. Energy predictions of PROFET are precise, with low error bars, and accurate, with small difference from the actual values measured on real hardware.

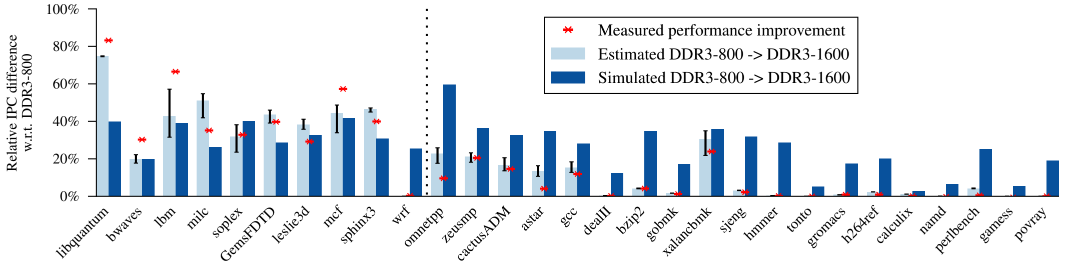


Fig. 14. Sandy Bridge, DDR3-800 → 1600: Comparison of the estimated (PROFET) and simulated (ZSim+DRAMSim2) performance improvement. Estimations of PROFET correspond to the real-system measurements much better than the simulated performance.

systems, and, to the best of our knowledge, it is one of the fastest system simulators available. Different main memory configurations were simulated with **DRAMSim2** [32]. DRAMSim2 is a cycle accurate model of the DDR3 main memory validated against manufacturer Verilog models.

The accuracy of PROFET and ZSim+DRAMSim2 were compared on an example of increasing memory frequency from DDR3-800 to DDR3-1066/1333/1600. To keep the simulation time at the acceptable level, we focus on the CPU2006 benchmarks and exclude the HPC scientific applications. In all the experiments we fully utilize the available 16 cores of the Sandy Bridge platform by executing 16 benchmark copies. The simulations were limited to 150 billion instructions of each benchmark. In order to make a fair comparison with the simulator, the benchmark execution on the actual system and the corresponding PROFET predictions were done for the same 150 billion instructions.

B.2.2 Results. In Figure 14, we compare the measured, simulated and estimated performance improvement when increasing the DRAM frequency from DDR3-800 to DDR3-1600. The DDR3-800→DDR3-1066/1333 results show the same trend and lead to the same conclusions.

The actual measured values (red cross markers) show significant performance improvement for the high-bandwidth benchmarks (left hand-side of the chart) and no performance changes for the low-bandwidth benchmarks (right hand-side of the chart). PROFET's estimations are accurate, with an average error of 3.6%, and closely follow the trend of the actual measurements. Also, for the benchmarks for which the estimation error is moderate, e.g., *lbm* and *milc*, the estimated performance have high error bars, clearly indicating a limited estimation precision in these cases. The simulated performance shows significant discrepancy with the actual measured values. The average simulation error is 15.7%, which is significant considering that the average DDR3-800→DDR3-1600 performance improvement is 17.5%. The range of the simulator error is also very high, between −23.7% (*libquantum*) and 45.7% (*omnetpp*). Finally, trend of the simulated results is completely different from the actual one. As already mentioned, high-bandwidth benchmarks experience high performance improvement, and insignificant performance changes for the low-bandwidth benchmarks. The simulator underestimates performance gains of the high-bandwidth benchmarks (left hand-side of the chart) while it overestimates gains for the low-bandwidth benchmarks (right hand-side of the chart). Therefore, the simulated performance gains are roughly uniform over the benchmark suite, which is a completely different trend from the actual measurements.

B.2.3 Time to perform prediction. We now compare the total time for prediction using PROFET against ZSim and DRAMSim2 hardware simulators. In our experiments, running the hardware simulators on a representative segment of the SPEC CPU2006 benchmark suite took 242 hours. Running the PROFET model required three steps. First, memory profiling to obtain 26 curves (0%

to 50% proportion of reads at 2% increments) with 35 points each, for DDR4, at a cost of 1 second per experiment, required 910 seconds. Second, profiling of the representative segments of the benchmark suite on the real machine required 127 seconds. Third, running the PROFET model required a couple of seconds. In total, PROFET required 1040 seconds, which is 838 times faster than the hardware simulators.

B.2.4 Discussion. In addition to the better accuracy, PROFET has various advantages over hardware simulators. PROFET is faster than the hardware simulators by almost three orders of magnitude, so it can be used to analyze production HPC applications, arbitrarily sized systems, and numerous design options. In this paper we presented experiments on two hardware platforms, Sandy Bridge and KNL. We analyzed four memory configurations for Sandy Bridge (DDR3-800/1066/1333/1600), and two for the KNL (DDR4-2400 and MCDRAM). In all the experiments, we analyzed all the benchmarks from the SPEC CPU2006 suite. Finally, for the Sandy Bridge platform, we also analyzed power and energy consumption in each memory configuration, and four HPC production applications. Performing the study of this size by using hardware simulators would be impossible within a practical length of time.

Additionally, the method is based on profiling of the application's memory behavior, so it does not require detailed modeling of the CPU as it already takes account of the real (and not publicly disclosed) data prefetcher and out-of-order engine. Therefore, it can be used to model various platforms as long as they support the required application profiling. PROFET was initially developed for the Sandy Bridge platform, and later we evaluated it for the KNL server. Adjustment of PROFET to the KNL system was trivial, as it required changes to only a few hardware parameters, such as, for example the reorder buffer size.

We release the PROFET source code as open source [31]. The release includes all PROFET inputs and outputs and evaluation results for the case study that is used in the rest of this paper. The package includes the memory system profiles, CPU parameters, application profiles and memory power parameters, as well as the power, performance and energy outputs from PROFET and the measurements on the baseline and target platforms. The released PROFET model is ready to be used on high-end Intel platforms, and we would encourage the community to evaluate PROFET versus actual hardware platforms and hardware simulators, and share their findings.

REFERENCES

[1] 2018. TOP500 List. http://www.top500.org/.

[2] Arira Design. 2013. Hybrid Memory Cube Evaluation & Development Board. http://www.ariradesign.com/hmc-board.

[3] Yuan Chou, Brian Fahs, and Santosh Abraham. 2004. Microarchitecture Optimizations for Exploiting Memory-Level Parallelism. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*. 76–87.

[4] R. Clapp, M. Dimitrov, K. Kumar, V. Viswanathan, and T. Willhalm. 2015. Quantifying the Performance Impact of Memory Latency and Bandwidth for Big Data Workloads. In *IEEE International Symposium on Workload Characterization*. 213–224. https://doi.org/10.1109/IISWC.2015.32

[5] Yokogawa Test & Measurement Corporation. [n.d.]. WT230 Digital Power Meter. https://cdn.tmi.yokogawa.com/IM760401-01E.pdf.

[6] S. Van den Steen, S. Eyerman, S. De Pestel, M. Mechri, T. E. Carlson, D. Black-Schaffer, E. Hagersten, and L. Eeckhout. 2016. Analytical Processor Performance and Power Modeling Using Micro-Architecture Independent Characteristics. *IEEE Trans. Comput.* 65, 12 (Dec 2016), 3537–3551. https://doi.org/10.1109/TC.2016.2547387

[7] Qingyuan Deng, David Meisner, Luiz Ramos, Thomas F. Wenisch, and Ricardo Bianchini. 2011. MemScale: Active Low-power Modes for Main Memory. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*. 225–238.

[8] P. G. Emma. 1997. Understanding some simple processor-performance limits. *IBM Journal of Research and Development* 41, 3 (May 1997), 215–232. https://doi.org/10.1147/rd.413.0215

[9] Stijn Eyerman, Lieven Eeckhout, Tejas Karkhanis, and James E. Smith. 2006. A Performance Counter Architecture for Computing Accurate CPI Components. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*. 175–184.

[10] Stijn Eyerman, Lieven Eeckhout, Tejas Karkhanis, and James E. Smith. 2009. A Mechanistic Performance Model for Superscalar Out-of-order Processors. *ACM Trans. Comput. Syst.* 27, 2 (May 2009), 3:1–3:37.

[11] Xixhou Feng, Rong Ge, and K. W. Cameron. 2005. Power and energy profiling of scientific applications on distributed systems. In *IEEE International Parallel and Distributed Processing Symposium*. https://doi.org/10.1109/IPDPS.2005.346

[12] D. Genbrugge, S. Eyerman, and L. Eeckhout. 2010. Interval simulation: Raising the level of abstraction in architectural simulation. In *The Sixteenth International Symposium on High-Performance Computer Architecture*. 307–318. https://doi.org/10.1109/HPCA.2010.5416636

[13] Andrew Glew. 1998. MLP yes! ILP no! *International Conference on Architectural Support for Programming Languages and Operating Systems, Wild and Crazy Ideas Session* (Oct. 1998).

[14] John L. Hennessy and David A. Patterson. 2017. *Computer Architecture: A Quantitative Approach* (6th ed.).

[15] Intel Corporation. 2012. *Intel® Xeon® Processor E5-1600/E5-2600/E5-4600 Product Families Datasheet - Volume One*. Technical Report 326508.

[16] Intel Corporation. 2012. *Intel® Xeon® Processor E5-2600 Product Family Uncore Performance Monitoring Guide*. Technical Report.

[17] Intel Corporation. 2016. *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. Technical Report.

[18] Intel Corporation. 2017. *Intel® Xeon Phi™ Processor Performance Monitoring Reference Manual - Volume 2: Events*. Technical Report.

[19] Bruce Jacob, Spencer Ng, and David Wang. 2007. *Memory Systems: Cache, DRAM, Disk*.

[20] Bruce L. Jacob. 2009. The Memory System: You Can't Avoid It, You Can't Ignore It, You Can't Fake It. *Synthesis Lectures on Computer Architecture* 4, 1 (2009), 1–77.

[21] James Jeffers, James Reinders, and Avinash Sodani. 2016. *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition* (2nd ed.).

[22] Tejas S. Karkhanis and James E. Smith. 2004. A First-Order Superscalar Processor Model. In *Proceedings of the Annual International Symposium on Computer Architecture*. 338–349.

[23] Y. Kim, W. Yang, and O. Mutlu. 2016. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE Computer Architecture Letters* 15, 1 (Jan. 2016), 45–49.

[24] Peter Kogge, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Robert Lucas, Mark Richards, Al Scarpelli, Steven Scott, Allan Snavely, Thomas Sterling, R. Stanley Williams, and Katherine Yelick. 2008. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems.

[25] David Kroft. 1981. Lockup-free Instruction Fetch/Prefetch Cache Organization. In *Proceedings of the Annual Symposium on Computer Architecture*. 81–87.

[26] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. 2009. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. 469–480.

[27] John D. McCalpin. 1991-2007. *STREAM: Sustainable Memory Bandwidth in High Performance Computers*. Technical Report. University of Virginia. http://www.cs.virginia.edu/stream/

[28] Micron Technology, Inc. 2007. *Calculating Memory System Power for DDR3*. Technical Report TN-41-01.

[29] Micron Technology, Inc. 2013. MT36JSF1G72PZ-1G6M1, 8GB (x72, ECC, DR) 240-Pin DDR3 RDIMM. http://www.micron.com/~/media/documents/products/datasheet/modules/parity_rdimm/jsf36c1gx72pz.pdf.

[30] Partnership for Advanced Computing in Europe (PRACE). 2013. Unified European Applications Benchmark Suite. www.prace-ri.eu/ueabs/.

[31] Milan Radulovic, Rommel Sanchez Verdejo, Paul Carpenter, Petar Radojković, Bruce Jacob, and Eduard Ayguadé. 2019. PROFET — Analytical model that quantifies the impact of the main memory on application performance and system power and energy consumption. https://github.com/bsc-mem/PROFET.

[32] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. 2011. DRAMSim2: A Cycle Accurate Memory System Simulator. *IEEE Computer Architecture Letters* 10, 1 (Jan. 2011), 16–19.

[33] Daniel Sanchez and Christos Kozyrakis. 2013. ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-core Systems. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*. 475–486.

[34] Rommel Sanchez Verdejo, Kazi Asifuzzaman, Milan Radulovic, Petar Radojković, Eduard Ayguadé, and Bruce Jacob. 2018. Main Memory Latency Simulation: The Missing Link. In *Proceedings of the International Symposium on Memory Systems*. 1–9.

[35] Avinash Sodani. 2011. Race to Exascale: Opportunities and Challenges. Keynote Presentation at the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO).

[36] A. Sodani, R. Gramunt, J. Corbal, H. S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y. C. Liu. 2016. Knights Landing: Second-Generation Intel Xeon Phi Product. *IEEE Micro* 36, 2 (March 2016), 34–46. https://doi.org/10.

1109/MM.2016.25

[37] Standard Performance Evaluation Corporation. [n.d.]. SPEC CPU 2006. http://www.spec.org/cpu2006/.

[38] Rick Stevens, Andy White, Pete Beckman, Ray Bair-ANL, Jim Hack, Jeff Nichols, Al GeistORNL, Horst Simon, Kathy Yelick, John Shalf-LBNL, Steve Ashby, Moe Khaleel-PNNL, Michel McCoy, Mark Seager, Brent Gorda-LLNL, John Morrison, Cheryl Wampler-LANL, James Peery, Sudip Dosanjh, Jim Ang-SNL, Jim Davenport, Tom Schlagel, BNL, Fred Johnson, and Paul Messina. 2010. A Decadal DOE Plan for Providing Exascale Applications and Technologies for DOE Mission Needs. Presentation at Advanced Simulation and Computing Principal Investigators Meeting.

[39] J. Treibig, G. Hager, and G. Wellein. 2010. LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments. In *International Conference on Parallel Processing Workshops*. 207–216. https://doi.org/10.1109/ICPPW.2010.38

[40] R. S. Verdejo and P. Radojković. 2017. Microbenchmarks for Detailed Validation and Tuning of Hardware Simulators. In *2017 International Conference on High Performance Computing Simulation (HPCS)*. 881–883.

[41] Wm. A. Wulf and Sally A. McKee. 1995. Hitting the Memory Wall: Implications of the Obvious. *ACM SIGARCH Computer Architecture News* 23, 1 (March 1995), 20–24.

Received October 2018; revised February 2019; accepted April 2019