Abstract

The ever growing complexity of high performance computing systems imposes significant challenges to exploit as much as possible their computational and communication resources. Recently, the Cache-aware Roofline Model has has gained popularity due to its simplicity modeling multi-cores with complex memory hierarchy, characterizing applications' bottlenecks, and quantifying achieved or remaining improvements. In this short paper we push this model a step further to model NUMA and heterogeneous memories with a handy tool, and spot data locality bottlenecks on such systems.

Keywords Roofline Model, heterogeneous memory, NUMA, Cache

I. Introduction

Emerging memory technologies (e.g heterogeneous memory architectures with non-volatile memory, on-package memory . . .), are required to address applications' needs and improve the performance at the cost of a growing hardware complexity. The memory wall makes memory management one of the main bottlenecks, and the variety expansion of allocable memories will definitely make data locality a performance hot spot, even on single chips.

We think we can leverage the Roofline Model to detect the latters accross on-chip's memories. For instance, a data allocated on low-bandwidth memory, may lead the application to become bandwidth bound. Our extension to heterogeneous and NUMA memory of the Cache Aware Roofline Model shows that in such sytems, memory bandwidth may vary from one pair (Core, memory) to another. Thus memory bound applications with bad data locality can be spotted by observing their performance variation beeing similar to memories performance differences.

In our contribution, we developped a handy tool to build and validate the model on such systems.

The remainder of this paper is organized as follow:

The section II will describe the Roofline Model and its main technical variations. Finally section III will briefly describe our implementation to measure memory bandwidths, and validate the model.

II. The Roofline Model Then and Now

The original paper [7] depict a machine with two subsystems: a memory and a compute unit, executing the instructions dispatched from a common instruction channel.

This idea is drawn on figure 1. Depending on the operational intensity (i.e. the ratio of compute instructions (blue rectangles) over memory instructions (green rectangles)), one unit, the other or both may be saturated with instructions. The width of the memory channel is the bandwidth and the width of the compute channel is the floating point peak (fpeak) performance. The model assumes there can be no dependency between instructions (they can overlap perfectly) and instructions' lattency is hidden.

On figure 2 is shown the graphical representation of the model. The operational intensity stands on abscissa and the performance stands on the ordinate axis. Top horizontal lines show the fpeak performance for different type of compute instructions and oblique lines show the memory bandwidth for different types of memories. On this representation one can see that: the roofline model ($min(bandwidth * operational\_intensity, fpeak)$) shows whether a

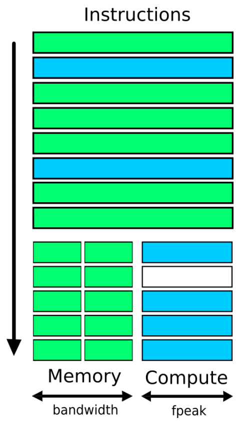

Figure 1: Roofline model draw

pattern of compute/memory instructions interleaving is either compute or memory bound if the maximum achievable performance is bound whether by the fpeak performance or the bandwidth.

The model has been successfully used in several [4] [5] [6] applications optimizations, whether to prove bottlenecks, or measure achievable (or achieved) improvements.

It has been applied to other memory subsystems [3], energy [2], and abstract runtime systems.

Based on the Cache Aware Roofline Model [3] methodology, we applied and validated the model to heterogeneous memory subsystems.

III. Contribution

In our contribution, we developped a tool able to benchmark rooflines and validate them with micro kernels, compatible with intel chips.

Each benchmark (rooflines, and validation) is made of assembly code using best architectures instructions available at compile time.

In brief, the fpeak benchmark repeats compute instructions and outputs $\frac{n*iFlops}{time}$ where n is the number of compute instructions and $iFlops$ is the number of floating point operations computed per instruction.

The bandwidth benchmark repeats coallesced memory accesses on a private[1] buffer of several sizes fitting the memory to benchmark

[1]Each thread have its own buffer's chunk

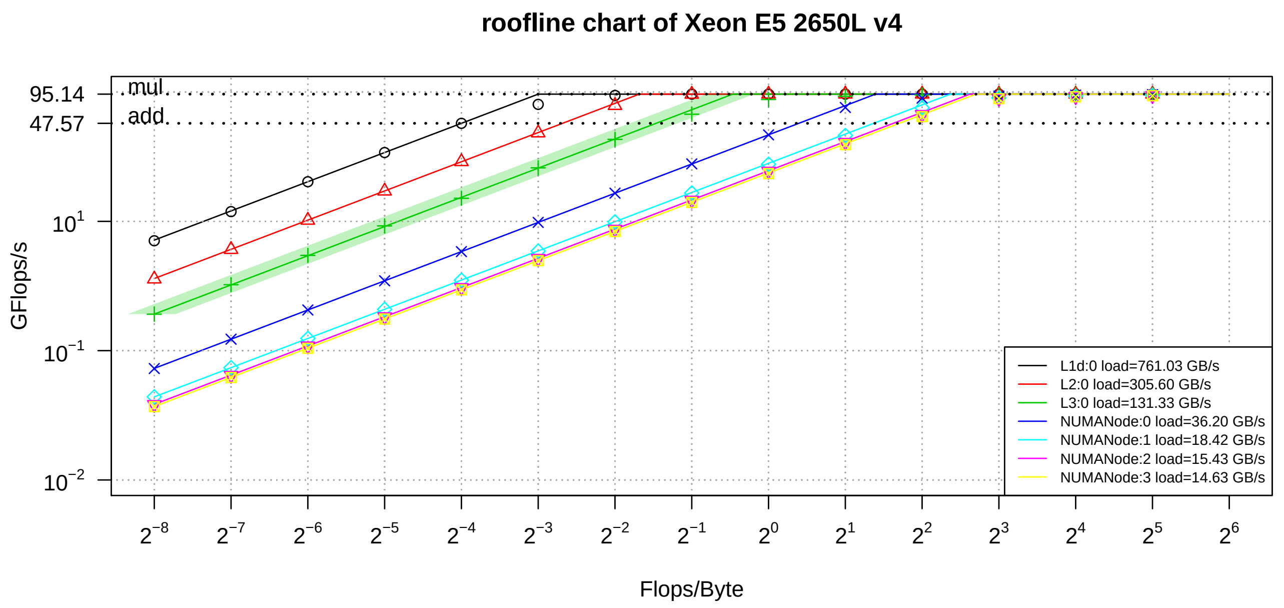

Figure 2: Example of the roofline chart with several bounds. Units are not instructions but rather, more standard unit, e.g flops for floating point operations (maybe several per instructions) and bytes transferred from memory (several per instruction).

and outputs $\frac{n*iBytes}{time}$ where n is the number of memory instructions and *iBytes* is the number of Bytes transferred per instruction.

And the validation kernels interleave previously defined benchmarks instructions and output its performance and operational intensity.

We can find several arithmetic floating point operation peaks (addition, multiplication, overlapping multiply add, and fuse multiply add), as well as several bandwidth types (load, store, non temporal load, non temporal store, interleaving of 2 loads and one store), and for several memory levels (L1, L2, L3, (caches . . .), NUMA memories, MCDRAM . . .) detectable with hwloc [1] library.

Figure 3 shows a model of such a machine given by hwloc. Nested boxes represent hierarchical inclusiveness of ressources in the machine. Our benchmark can work both in sequential and in parallel. In the former, one benchamrk thread is spawn on first Core (left most) of the machine. In the latter, one benchmark thread per Core of the first NUMA memory node is spawn, and threads measure (Flops and Bytes) are accumulated. The parallel model representation obtained from machine 3 is shown on figure 2. The system yields a load bandwidth of 761GB/s for the total of its 7 L1 caches per Node, whereas the shared local memory (NUMANode:0) yields a bandwidth of 36GB/s for the same type of micro-operations. Remote memories' bandwidth are obtained by running the same benchmark as local memory, but explicitly allocating the data on remote memories. On figure 2, the lines shows the top measures. One can notice, on L3 a wider line with less opacity showing the measured bandwidth deviation. For each memory we use different sizes of buffers fitting the memory, and depending on the buffer size, the bandwidth may vary. We can also notice that for L1 (black line), the points does not stick very well to lines around the ridge. Actually the points does not measure the bandwidths, but instead validate rooflines. They consist of micro-benchmarks of different known operational intensities, interleaving memory and compute instruction to reach rooflines, and show how realistic the model is with those metrics, by measuring each micro-benchmark's performance.

IV. Conclusion and future work

In this short paper we prestented the Roofline Model and an extension to heterogeneous memory's sytem using the Cache Aware Roofline Model methodology. Later on we will validate the interest of this representation with applications.

References

[1] B. Goglin. Managing the topology of heterogeneous cluster nodes with hardware locality (hwloc). In *High Performance Computing Simulation (HPCS), 2014 International Conference on*, pages 74–81, July 2014.

[2] A. Ilic, F. Pratas, and L. Sousa. Beyond the roofline: Cache-aware power and energy-efficiency modeling for multi-cores. *IEEE Transactions on Computers*, PP(99):1–1, 2016.

[3] Aleksandar Ilic, Frederico Pratas, and Leonel Sousa. Cache-aware roofline model: Upgrading the loft. *IEEE Computer Architecture Letters*, 13(1):21–24, 2014.

[4] Ki-Hwan Kim, KyoungHo Kim, and Q-Han Park. Performance analysis and optimization of three-dimensional {FDTD} on {GPU} using roofline model. *Computer Physics Communications*, 182(6):1201 – 1207, 2011.

[5] Diego Rossinelli, Christian Conti, and Petros Koumoutsakos. Mesh–particle interpolations on graphics processing units and multicore central processing units. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 369(1944):2164–2175, 2011.

[6] Rob V. van Nieuwpoort and John W. Romein. Using many-core hardware to correlate radio astronomy signals. In *Proceedings of the 23rd International Conference on Supercomputing*, ICS '09, pages 440–449, New York, NY, USA, 2009. ACM.

[7] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, April 2009.

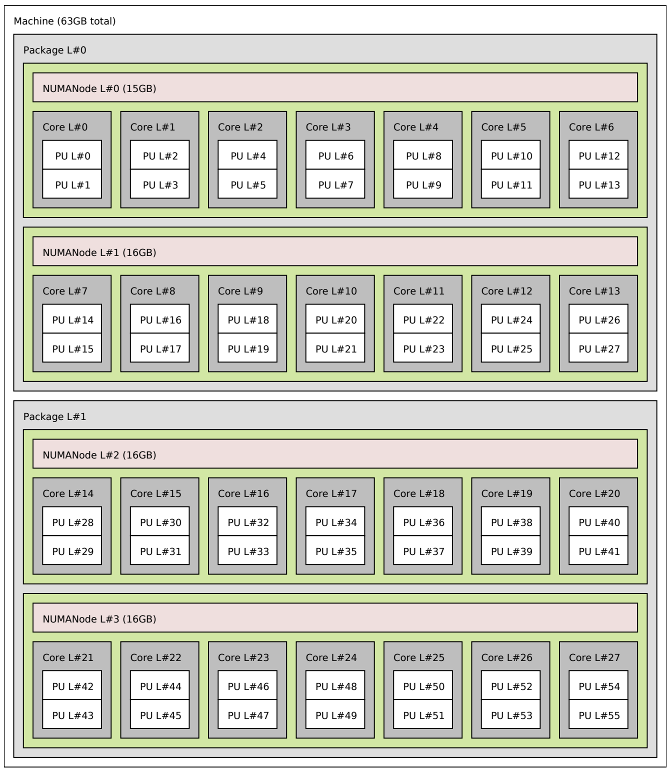

Figure 3: Machine topology, with hidden caches