informatik
die zukunft

Hausarbeit im Seminar:
Neueste Trends im
Hochleistungsrechnen

Performance Modeling

Supervisor: Dr. Julian Kunkel

Ahmed Hassan

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

WiSe 2016

Ahmed Hassan
Matr. Nr. 666 7124
4hassan@informatik.uni-hamburg.de

Contents

1. Introduction

Nowadays we are living in a vibrate world where everything is developed so fast. The capacity and the speed of the CPUs as well as memory chips are doubled every five years. A long with the fast developed technologies there is more focus on using the available technology more efficiently and to optimize it to match our needs.

As a result of such new trend there is more demand on what so called "Performance Modeling", where a model for the desired system is built before starting to optimize this model to match the desired performance then implementing it.

1.1. Motivation

When building a supercomputer to run a certain applications, it is always better to develop those applications based on a model for this supercomputer then starting to optimize both the software as well as the hardware when building this supercomputer. By doing that, a high optimized performance is achieved on a cheap cost without wasting a lot of money, time and power on optimization on a real system.

1.2. Structure of this Paper

This paper is divided into seven chapters. The first chapter provides a quick introduction about the performance modeling followed by a clear explanation about the performance modeling prospective and the factors affecting it in the second chapter. Moreover, the second chapter explains the difference between various performance models and how the performance components are integrated with each other's. In chapter three the roofline model is in question. Furthermore, a real example where roofline model is used to optimize the performance is also discussed in the third chapter, while the fourth chapter shows a glimpse view of Execution-Cache-Memory which is considered to be as a refinement to the roofline model. A quick introduction about the roofline model in HPC was introduced in the fifth chapter. Additionally, a glance overview of other levels of optimization is also discussed in the sixth chapter. The seventh and the final chapter summarize the topic of this paper.

2. Performance Modeling

2.1. What is Performance Modeling?

Performance modeling is a structured and repeatable approach to model the performance of a system that is by defining an abstract architectural model or software to simulate this model. It begins during the early phases of any system/application design and continues throughout the system/application life cycle.

2.2. Why Performance Modeling?

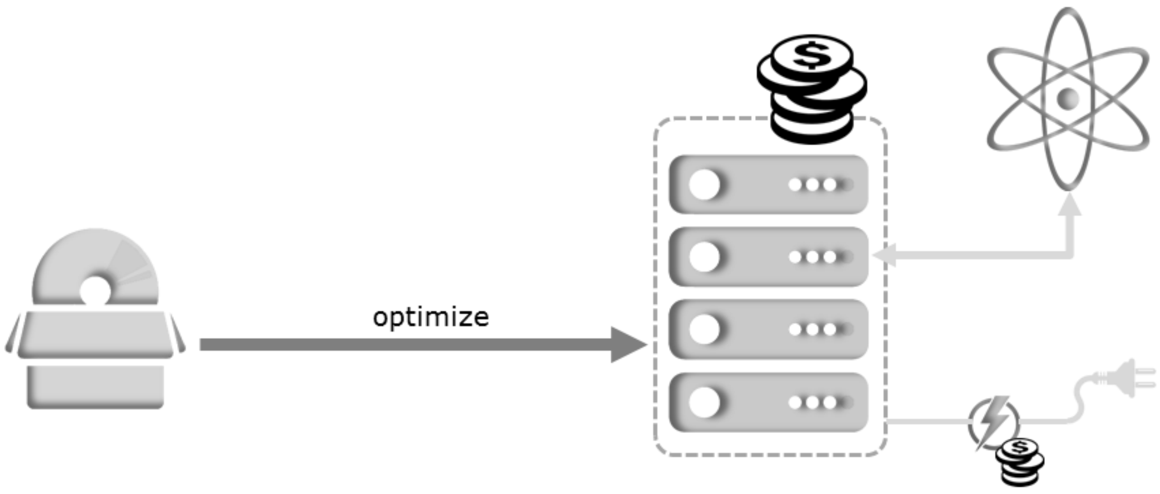


Figure 2.1: Performance modeling illustration

Let us assume that it is required to optimize applications to run on a multi-hundred-million dollar supercomputer that consumes as much energy as a small European town to solve computational problems at an international scale and advance science to the next level with “hero-runs” of scientific applications that cost $10 K and more per run. So, it is always better to plan ahead [1]. This is achieved by trying to extrapolate the performance from small machines to big one (see figure 2.1).

Parallel application performance is complex which is often unclear how the optimizations impact performance, especially at scale of different architectures at different scale and at different architect with those numbers of different parameters. So the goal is actually assessing the performance of the large scale application. Moreover, another problem is how to optimize for a machine that does not exist because the optimized machine is going to run for couple of years. If the application is not ready when the machine is deployed, then a lot of money is going to be wasted.

This means it is better to optimize for the application in the process or in parallel with deploying a machine, which means that the application cannot be ran on the machine looking for the performance, optimize it, running it again, optimize it, and so on because when this cycle is done, half of the life-time of the machine as well as a lot of compute power is wasted this way.

This is considered to be a big issue for applications that ran on large-scale systems where guided optimization is needed. So a good understand of the performance characteristic of the running machine before the deployment and guiding this optimization in parallel with the system is the best approach.

An alternative way is to use some hardware accelerators but this solution does not pay any attention to the bandwidth cost of such a technique so it is better to understand the system before making any trade-off in design.

So to sum up, Performance modeling provides several important benefits:

- Guide optimization during application design.
- Evaluate tradeoffs before building the solution.
- Guiding future maintenance and expansion decisions.
- Avoid performance surprises during application execution.
- Provide a document of itemized scenarios for tracking performance goals.

2.3. Prospective of Performance Modeling?

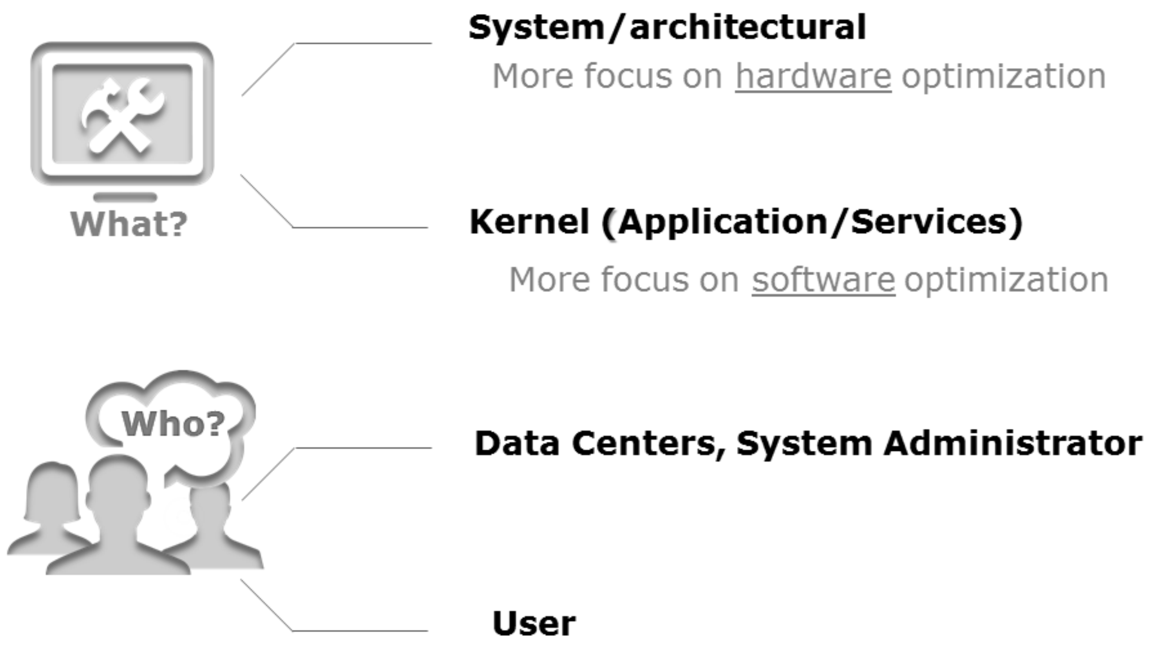

What?
System/architectural
More focus on hardware optimization
Kernel (Application/Services)
More focus on software optimization
Who?
Data Centers, System Administrator
User

Figure 2.2: Performance modeling prospective

Performance modeling can be implemented on System/architectural level where the focus will be more on hardware optimization or on the kernel (application/services) level where the optimization will be carried only on the software side. Having both sides mean that the audience of the performance modeling are data centers, system administrators as a backend system/software developers or system administrators. On the other side there is the end user who is going to use this system/software (see figure 2.2).

2.4. Factors Affecting System Performance

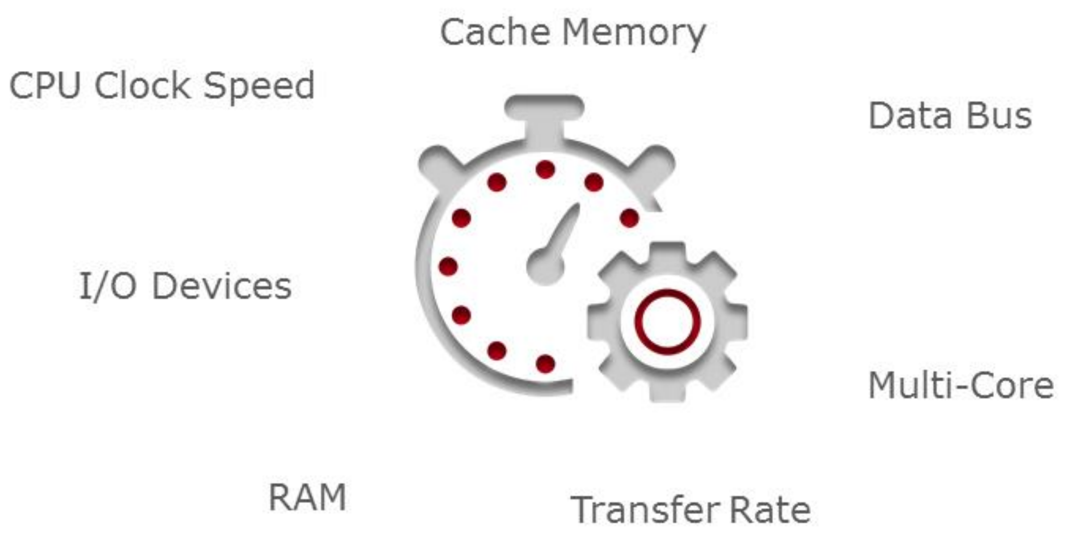


Figure 2.3: factors affecting system performance

There are a lot of factors that should be taken into consideration when optimizing the performance. Some of those factors are CPU clock speed, cache memory, data bus; multi-core CPU, RAM and I/O connected devices as well as transfer rate (see figure 2.3).

2.5. Principle Components of Performance [2]

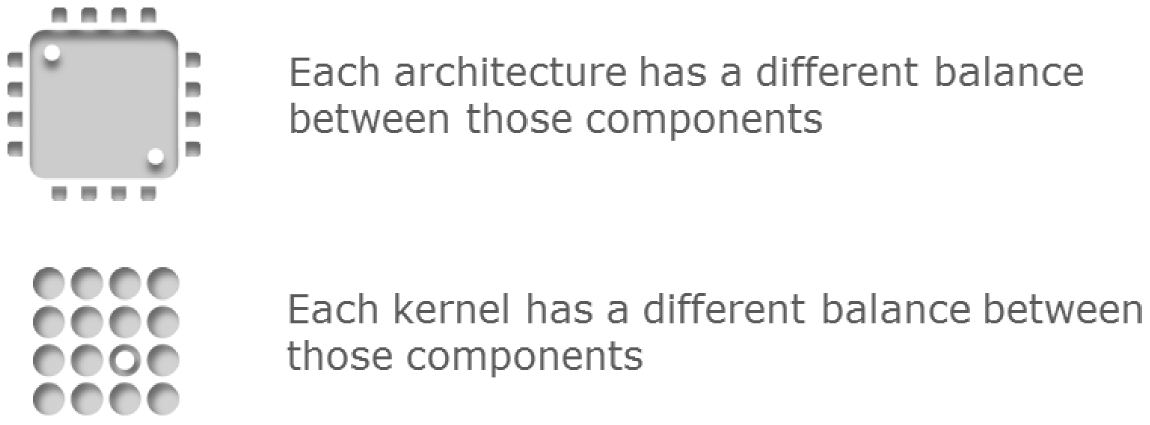

Each architecture has a different balance between those components
Each kernel has a different balance between those components

Figure 2.4: Principle components of performance

There are three principal components of performance [3]:

1. Computation
2. Communication
3. Locality

Note: each hardware architecture has a different balance between these three components as well as there is a different balance between those components on the kernel level (see figure 2.4). Performance is a question of how well a kernel's characteristics map to architecture's characteristics

1. Computation[4]

   The floating point performance (Gflop/s) is considered to be the main interest. The Peak in-core performance can be achieved through:

   - fully attainment Instruction-level parallelism (ILP), Data level parallelism (DLP), FMA;
   - non-FP instructions don't deplete instruction BW;
   - branch mis-predictions are not often;
   - Threads converge.

   On the other hand, In-core parallelism is achieved if:

   - The used algorithm implements Inheritance;
   - The generated code has explicit.

   Double-precision floating-point format is a computer number format that occupies 8 bytes (64 bits) in computer memory

   Note:
   Gflops: floating-point operations per second = (CPU Clock in GHz) × (Number of CPU Kernels)

2. Communication [4]

   DRAM bandwidth (GB/s) is the main interest. Peak bandwidth can be achieved when specific optimizations are implemented:

   - SW Prefetching;
   - NUMA allocation;
   - NUMA usage;
   - Memory coalescing;
   - Few unit stride streams.

3. Locality [4]

   Locality shows the location where the data is being processed. This means that, the traffic represents the volume of data to/from memory but not the number of loads and stores. Additionally, to reduce the communication it is better to increase the locality but there is still what so called "Compulsory Traffic" which is the minimum needed amount of communication that cannot be avoided.

   Locality optimization can be achieved on either the hardware level or on the software level. On the hardware level the modification of the hardware helps to reduce the communication. Such a hardware modification can be achieved through:

   - More cache associativities;
   - Increasing non-allocating caches;
   - Reduce capacity misses by increase cache capacities.

On the other hand, the software modification can be achieved by reducing the communication which is going to be done through:

- Avoid capacity misses through blocking;
- Avoiding conflict misses through Padding;
- Increasing non-allocating stores.

2.6. Integrating Performance Components

As the performance is considered an equation of the three mentioned components, the question now is how to integrate those three components to optimize the performance (see figure 2.5).

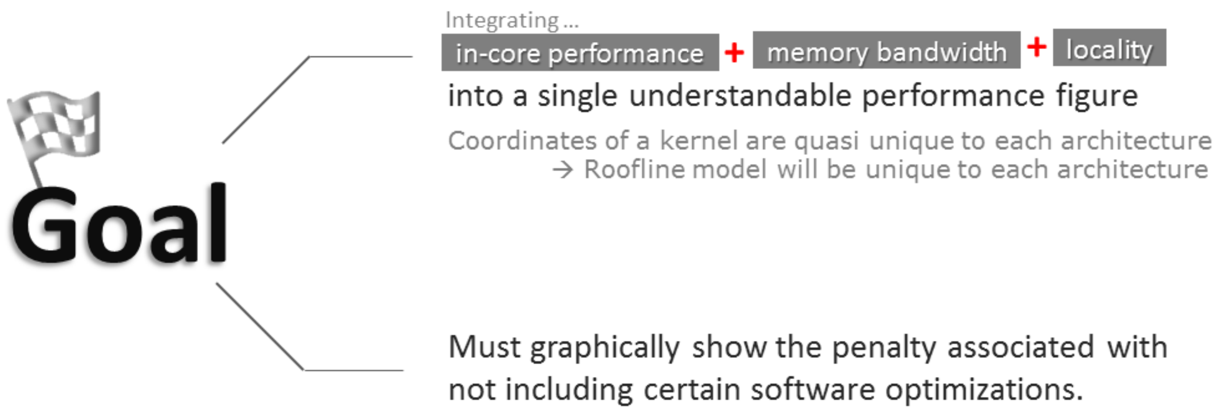

Goal
Integrating ...
in-core performance + memory bandwidth + locality
into a single understandable performance figure
Coordinates of a kernel are quasi unique to each architecture
→ Roofline model will be unique to each architecture
Must graphically show the penalty associated with not including certain software optimizations.

Figure 2.5: Integrating Performance Components

As the Coordinates of a kernel are unique to each architecture, then the performance optimization will be unique to each architecture. Flops: Bytes is the parameter that allows us to convert bandwidth (GB/s) to performance (GFlop/s) this will be achieved through Arithmetic Intensity and this can be achieved through measurements. By measuring the total bytes, all cache behavior (Compulsory misses, Capacity misses, Conflict misses) and Locality can be incorporated (see figure 2.6) [5].

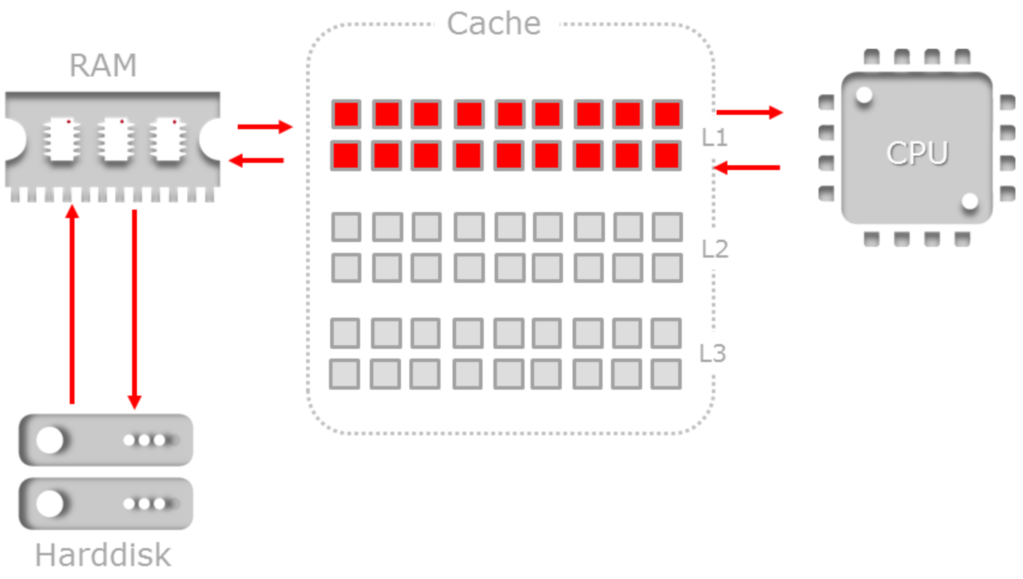


Figure 2.6: Caches Behavior

Cache behavior has three different types:

1. Compulsory misses

   Misses caused by the first reference to a location in memory that a program has never requested before.

2. Capacity misses

   Misses that occur regardless of associativity or block size, solely due to the finite size of the cache (pencil and paper or maybe performance counters).

3. Conflict misses

   Misses that could have been avoided, had the cache not evicted an entry earlier.

2.7. Different Performance Models

For system/architectural Performance Modeling, we have the following models

1. Stochastic Analytical Model
2. Statistical Performance Model
3. Roofline Model

Each model has its own advantages and disadvantages. For example, both stochastic analytical and statistical performance models can predict program performance on multiprocessors accurately. However, they rarely provide insights into how to improve performance of programs, compilers, or computers or they can be hard to use by non-experts [6].

On the other hand, Roofline model has a lot of advantages compared to the early mentioned two models. For example Roofline model provide a simpler approach to bound and bottleneck analysis. Furthermore, it provides valuable insight into the primary factors affecting the performance of computer systems. In particular, the critical influence of the system bottleneck is highlighted and quantified. Moreover, roofline model is also applicable to heterogeneous multicore computers. Nonetheless, roofline model ignores potentially important factors like block size, block allocation policy, and block replacement policy (see figure 2.7).

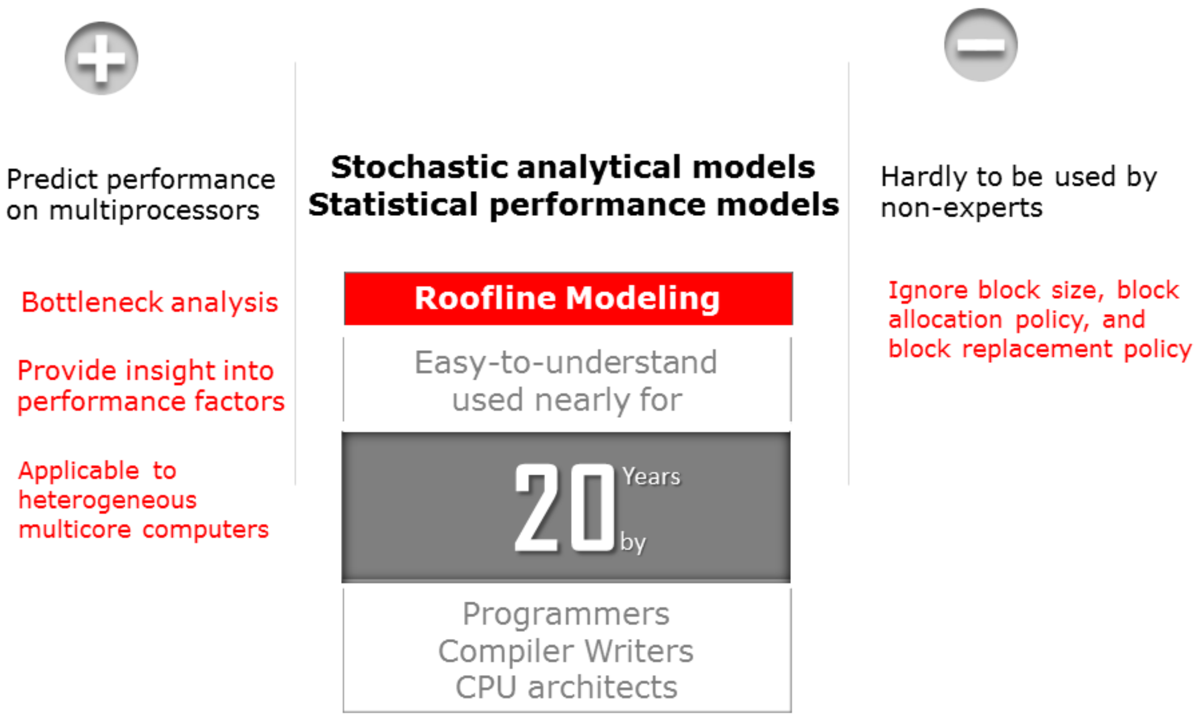

Predict performance on multiprocessors
Bottleneck analysis
Provide insight into performance factors
Applicable to heterogeneous multicore computers
Stochastic analytical models
Statistical performance models
Roofline Modeling
Easy-to-understand used nearly for
20 Years by
Programmers
Compiler Writers
CPU architects
Hardly to be used by non-experts
Ignore block size, block allocation policy, and block replacement policy

Figure 2.7: Pros and Cons of different Performance Models

Due to the its easy-to-understand, this model has been popular for nearly 20 years because it offers insights into the behavior of programs, helping programmers, compiler writers, and architects improve their respective designs.

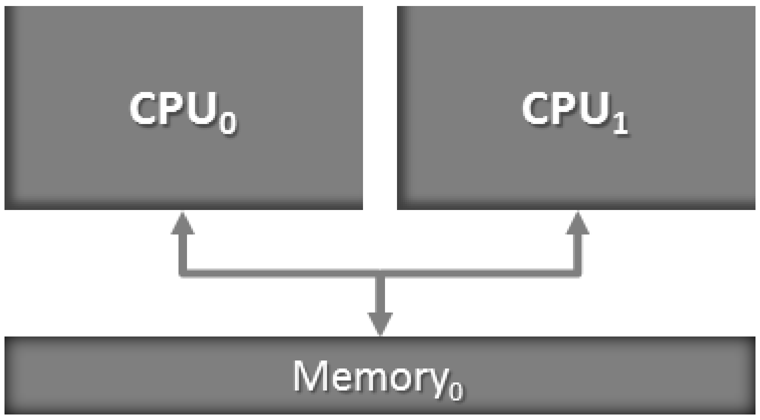
3. Roofline Model

3.1. Goals of Roofline Model [7]

The goals of Roofline model can be summarized in the following points:

- Provide a visually intuitive performance model
- Drive programmers towards better understanding of performance on modern computer architectures
  It does not only provide programmers with realistic performance expectations but also specifies potential challenges to performance
- Drive programmers to implement particular classes of optimizations by knowing of system bottlenecks.
- Focus on architecture-oriented roofline models (not using performance counters)

3.2. Explanation of Roofline Model [7]



Source: http://crd-legacy.lbl.gov/~oliker/papers/blw10_chapter_autotune.pdf, page 13

Figure 3.1: Simple Kernel Structure

Let us assume simple kernels that do the following processes:

1. Transfer Bytes of data from $Memory_0$
2. Perform F/2 FLOPs on both CPUs
3. Memory can support Peak Bandwidth Bytes/sec
4. The two CPUs combined can perform Peak Performance FLOPs/sec

P_{max}: Loop peak performance taking in consideration the data transfer from L1 cache (not necessarily P_{peak})

Computational intensity (I): “work” per byte transferred through the slowest data path (“the bottleneck”) (measured in flops/bytes). [6]

Code balance (B_c) = I^{-1}

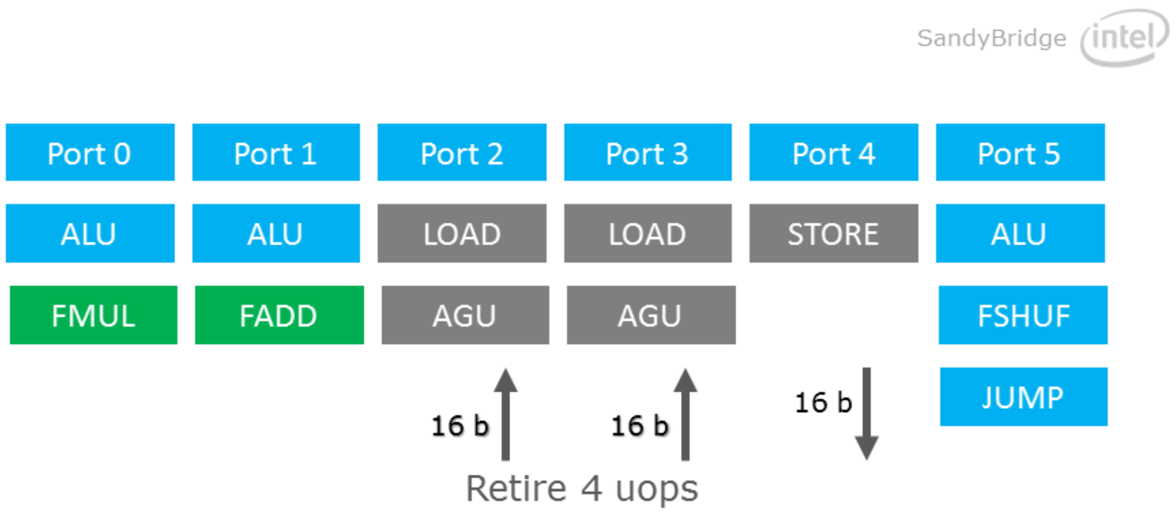
b_s: Peak bandwidth of the slowest data path (byte/sec)

Thus the Expected Performance will be the minimum of both P_{max} and $I.b_s$

$$P = \min (P_{max}, I \cdot b_s)$$

3.3. Roofline Model Analysis for Intel Sandy Bridge [8] [9]

In the following example Intel® SandyBridge™ is going to be used as an example to apply roofline model analysis on it. Assume also that all instructions in a loop are maintained independently to different ports and the sum number of penalty cycles for each cycle with Advanced Vector Extensions (AVX) (see figure 3.2).



Source: http://crd.lbl.gov/assets/pubs_presos/parlab08-roofline-talk.pdf, page 5

Figure 3.2: Intel SandyBridge Internal Structure

- one AVX MULT + one AVX ADD;
- One load instruction + ½ store instruction.

Per cycle with SSE or scalar

- Two load instruction;
- One MULT + one ADD instruction.

There is maximum of four micro-ops but considering three is more realistic.

Assuming the following code:

```
Double *A, *B, *C, *D;
For (int i=0; i<N; i++) {
A[i] = B[i] + C[i] * D[i]}
```

The number of cycles to process one AVX-vectorized iteration will be as following:

Cycle 1: LOAD + ½ STORE + MULT + ADD
Cycle 2: LOAD + ½ STORE
Cycle 3: LOAD

As one AVX iteration (3 cycles) performs 4 x 2 = 8 Flops -> 8 Flops / 3 cy
3 Gcy/s * 8 F / 3 cy = 8 GFlops/s

Bandwidth Calculation:
8 GFlops/s * 32 Byte / 2 Flops = 128 GBytes/s
Assume 3 GHz 8-core Sandy Bridge chip
$\mathbf{b_s}$ = 40 GB/s

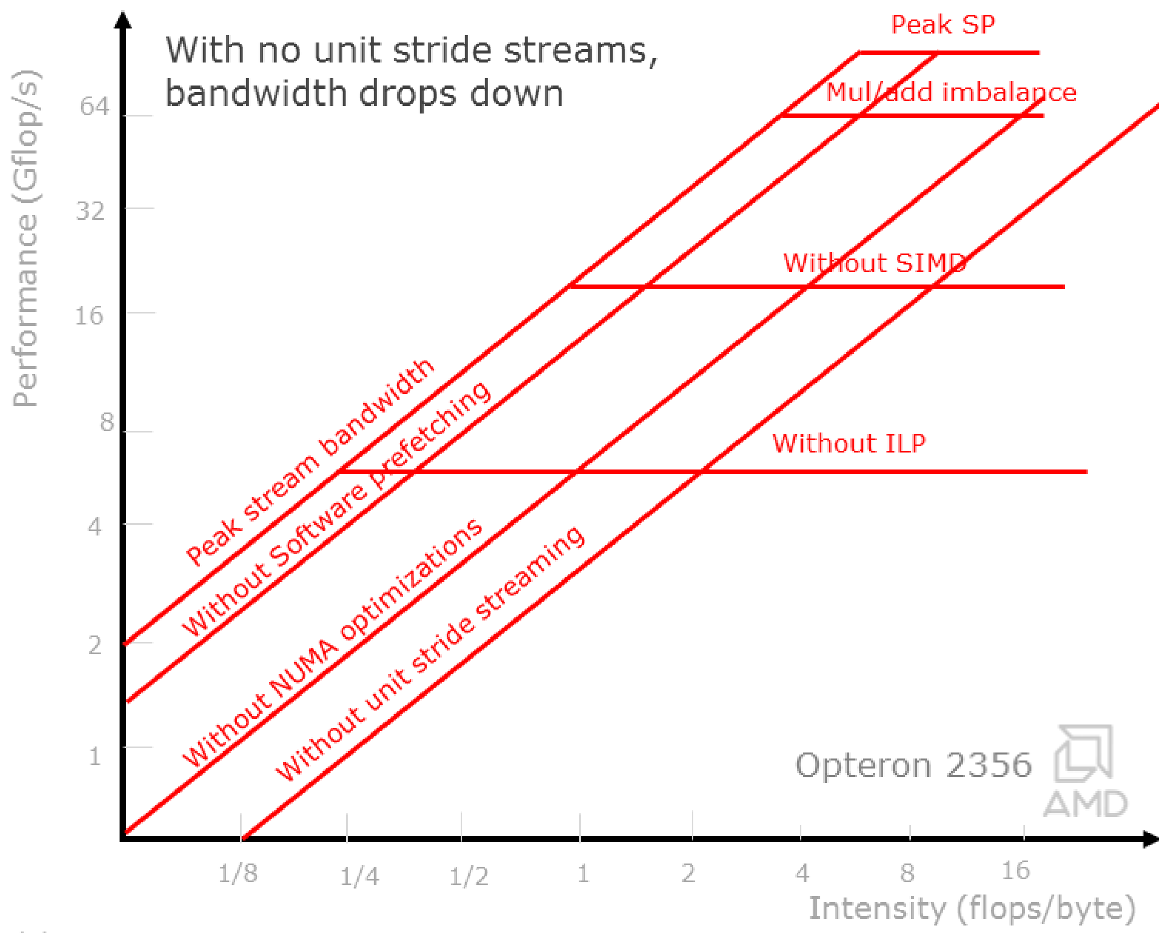
B_c = (4+1) Words / 2 Flops = 2.5 W/F
I = 0.4 F/W = 0.05 F/B
$I \cdot b_s$ = 2.0 GF/s (1.04 % of peak performance)

P_{peak} = 192 Gflop/s (8 cores x (4+4) Flops/cy x 3.0 GHz)
P_{max} = 8 x 8 Gflop/s = 64 Gflop/s (33% peak)

Now, taking the minimum of both P_{max} and $I.b_s$

$P = \min(P_{max}, I.b_s)$ = min (64,2.0) GFlop/s = 2.0 GFlops

For simplicity and diversity, AMD® Opteron™ 2356 (Barcelona) will be considered to show the optimization options and how each factor is going to affect the optimization graph of the AMD® Opteron™ processor (see figure 3.3) [10].



Source: http://crd.lbl.gov/assets/pubs_presos/parlab08-roofline-talk.pdf, page 21

Figure 3.3: Optimization graph for AMD® Opteron™ 2356

Note: On log-log scale, line always appears at a 45-degree angle.

From the previous graphs we notice that

- There is no standard/single ordering or roofline model.
- The ceiling order is generally bottom up.
- Addition, Multiplication and FMA are balanced inherent in many linear algebra routines.
- Addition is the most dominant operation thus the multipliers and FMA go underutilized.
- Processor can access its local memory faster than non-local or shared memory
- Memory access time depends on memory location relative to processor

As Instruction-Level parallelism (ILP) measures the number of operations that a program can be performed simultaneously. If the thread of execution falls short of expressing this degree of parallelism, functional units will go idle, and performance will be negatively affected.

Prefetching occurs when a processor requests an instruction from main memory before it is actually needed. Once the instruction comes back from memory, it is placed in a cache. When an instruction is actually needed, the instruction can be accessed much more quickly from the cache than if it had to make a request from memory.

4. Execution-Cache-Memory (ECM)

Execution-Cache-Memory (ECM) model refines the well-known roofline model, since it can predict the scaling and the saturation behavior of bandwidth-limited loop kernels on a multicore chip [11].

- Refines the roofline model to predict the behavior of saturation and scaling of bandwidth-limit.
- Provides a clear understand of the single and multi-core performance of streaming kernels.

Execution-Cache-Memory (ECM) model describes the single and multi-core performance of streaming kernels [11] [12].

Intel© SandyBridge™ processor exposes part of its power characteristics to the programmer through "Running Average Power Limit" feature.

- The majority of numerical codes are based on streaming loop kernels.
- Kernels are always limited by the bandwidth of memory that leads to a distinct scaling behavior across the cores of a multicore chip [13].

Roofline model is used to anticipate the performance

BUT

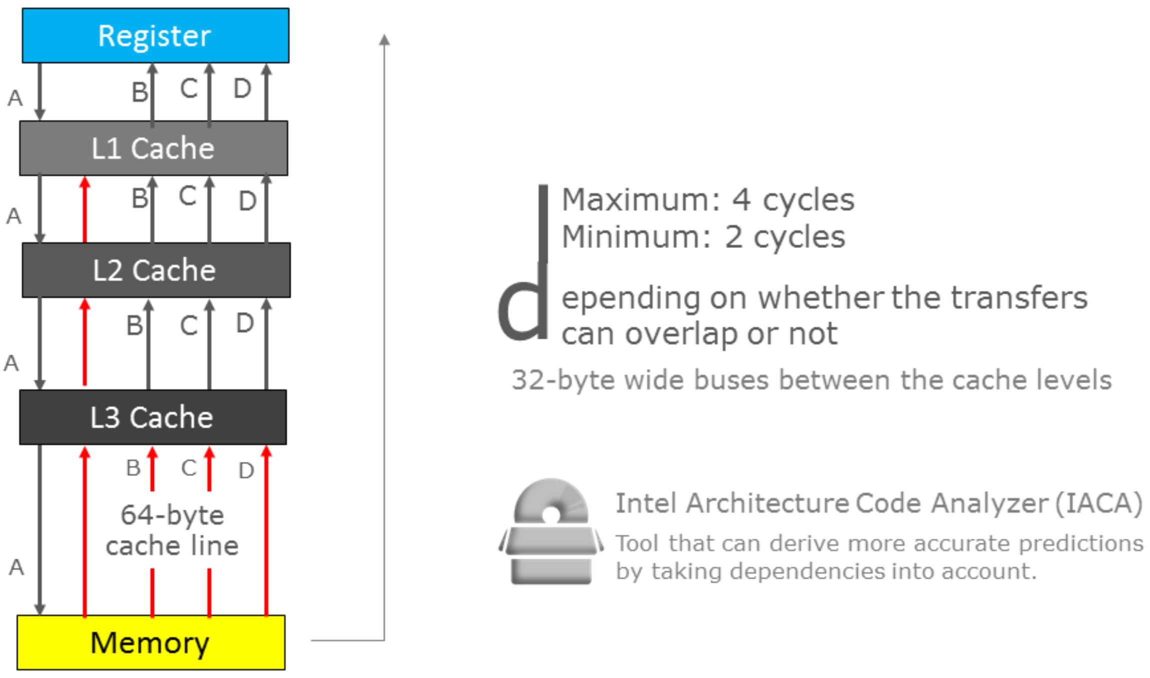
ECM model provides essential insight about the cache bandwidths and organization on the multicore chip to show up a more authentic characterization on the single-core level.

Execution-Cache-Memory (ECM) composes of [20]:

1. Core Time (T_{core}):
   Time taken to execute all instructions, with all operands of loads/ stores coming from/ going to the L1 data cache.

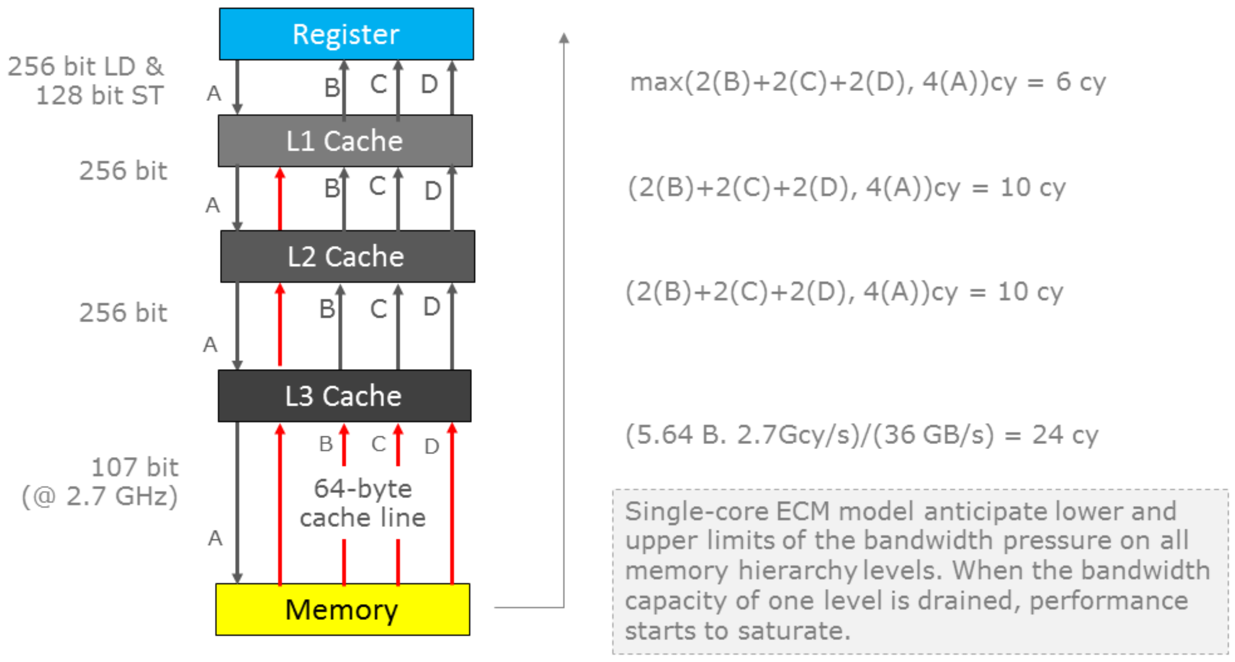
2. Data Delay (T_{data}):
   Time taken to transfer data to/ from L1 through the memory hierarchy. This value will be larger if the required cache line(s) are "far away".

The ECM analysis of a single core processor is as shown below (figure 4.1), while the ECM analysis for multicore scaling is as shown in (figure 4.2) as well as (figure 4.3) [17] [19].


Register
A
B
C
D
L1 Cache
L2 Cache
L3 Cache
64-byte cache line
Memory
Maximum: 4 cycles
Minimum: 2 cycles
depending on whether the transfers can overlap or not
32-byte wide buses between the cache levels
Intel Architecture Code Analyzer (IACA)
Tool that can derive more accurate predictions by taking dependencies into account.

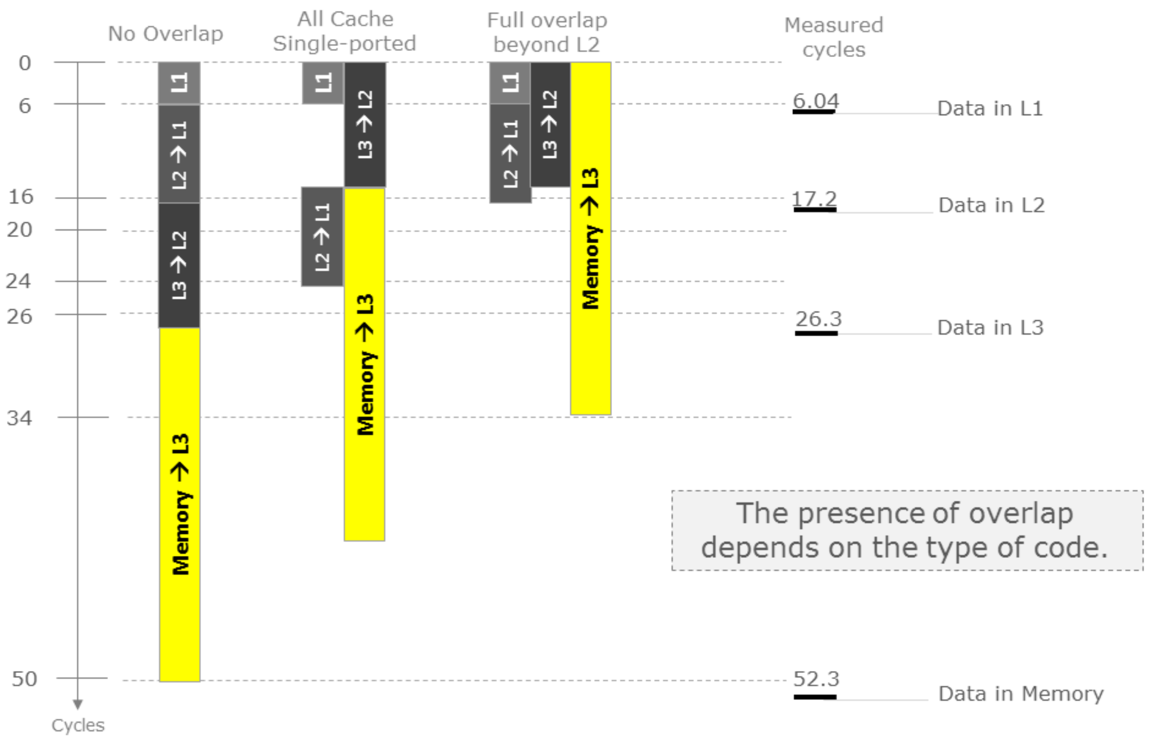
Source: http://moodle.rrze.uni-erlangen.de/mod/resource/view.php?id=3976, page 6

Figure 4.1: ECM Analysis for Single core


256 bit LD & 128 bit ST
256 bit
256 bit
107 bit (@ 2.7 GHz)
Register
A
B
C
D
L1 Cache
L2 Cache
L3 Cache
64-byte cache line
Memory
max(2(B)+2(C)+2(D), 4(A))cy = 6 cy
(2(B)+2(C)+2(D), 4(A))cy = 10 cy
(2(B)+2(C)+2(D), 4(A))cy = 10 cy
(5.64 B. 2.7Gcy/s)/(36 GB/s) = 24 cy
Single-core ECM model anticipate lower and upper limits of the bandwidth pressure on all memory hierarchy levels. When the bandwidth capacity of one level is drained, performance starts to saturate.

Source: http://moodle.rrze.uni-erlangen.de/mod/resource/view.php?id=3976, page 6

Figure 4.2: ECM Analysis for Multicore Scaling

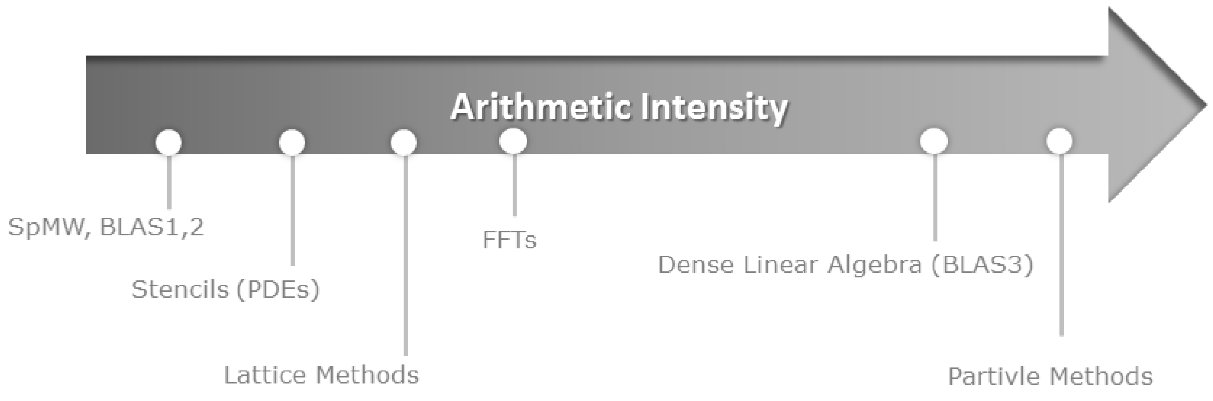


Source: http://moodle.rrze.uni-erlangen.de/mod/resource/view.php?id=3976, page 7

Figure 4.3: ECM Analysis for Multicore Scaling

As shown in figure (4.3) the performance of the processor is drastically improved by using full overlap beyond L2 compared using only single –ported as well as no overlap [18] [19].

5. Roofline in HPC Overview



Source: https://redmine.scorec.rpi.edu/attachments/111/roofline_for_FastMath.pdf, page 4

Figure 5.1: Roofline model for HPC

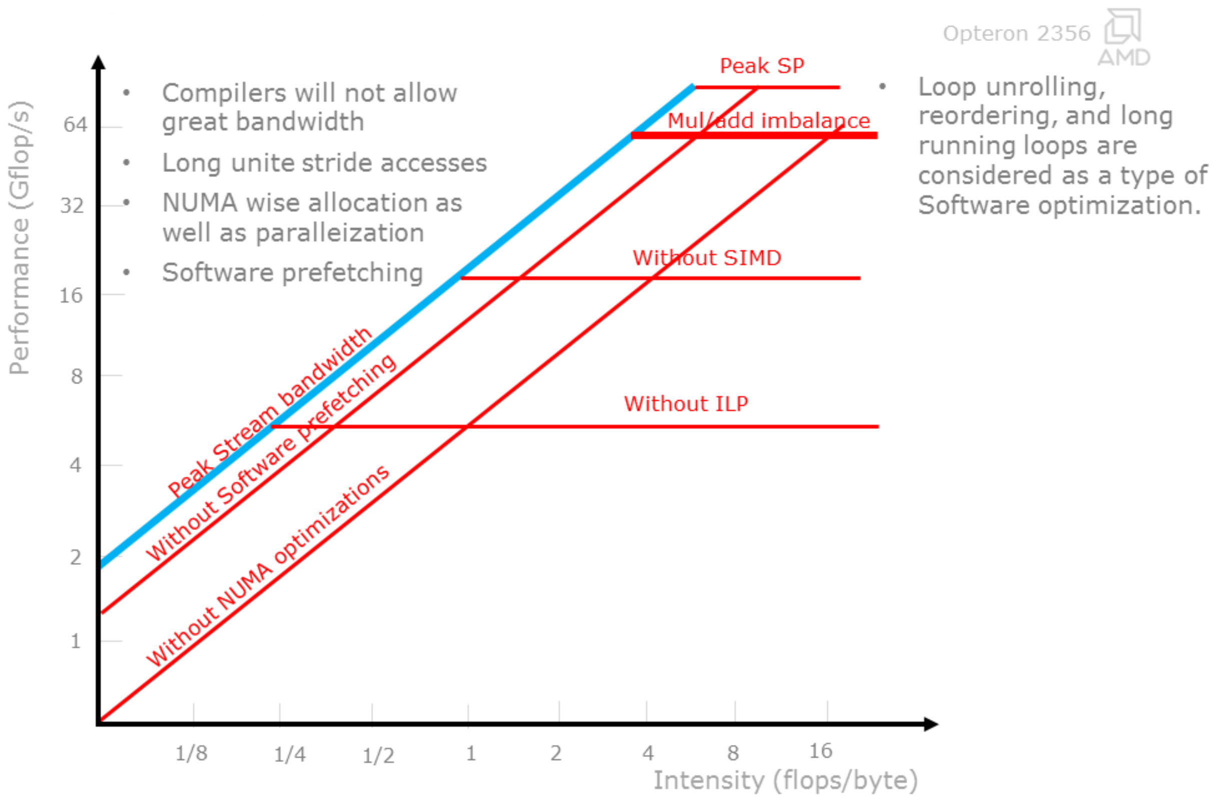
As shown in the figure 5.1 the following points are recommended after applying Roofline model on HPC [10]:

- Certain arithmetic intensity is exceeding by local store space.
- Arithmetic Intensity (AI) ~ Total Flops / Total DRAM Bytes
- Some HPC kernels have a constant arithmetic intensity.

6. Software/Service Optimization

6.1. Software Optimization

Considering software optimization for AMD® Opteron™ 2356 processor there are a lot of recommendations that will help to improve the performance of the processor such as NUMA wise allocation and long unite stride as well as software prefetching (see figure 6.1) [10] [16].



Source: http://crd.lbl.gov/assets/pubs_presos/parlab08-roofline-talk.pdf, page 20

Figure 6.1: Software Optimization

6.2. Application/Service Modeling

Other way of performance modeling is to model a communication between application/services [14].

Inputs

- Scenarios and design documentation about critical and significant use cases;
- Application design and target infrastructure and any constraints imposed by the infrastructure;
- QoS requirements and infrastructure constraints, including service level agreements (SLAs);
- Workload requirements derived from marketing data on prospective customers.

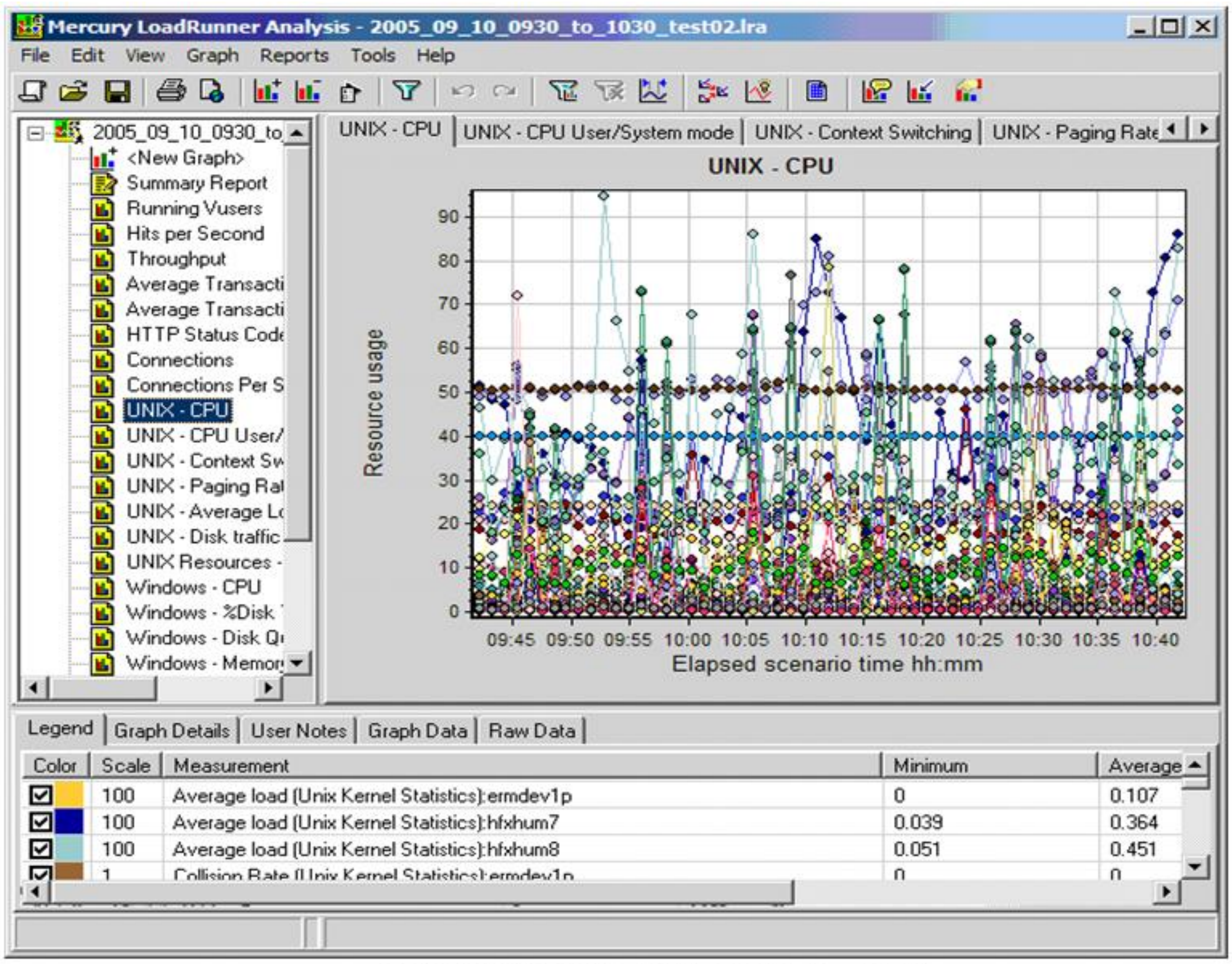
Outputs

- A performance model document;
- Test cases with goals.

For example if we are developing an online booking system then we measure the performance of the system with respect to our pre-defined SLA (see figure 6.2) [14] [20].
e.g.

- Number of Co-current booking requests;
- Number of Running Vusers;
- Number of Hit per Second;
- CPU etc.

Tools:
Commercial Tools like HP LoadRunner® or Open-source Tool like JMeter®.



Color	Scale	Measurement	Minimum	Average
☑ 🟨	100	Average load (Unix Kernel Statistics):ermdev1p	0	0.107
☑ 🟦	100	Average load (Unix Kernel Statistics):hfxhum7	0.039	0.364
☑ 🟦	100	Average load (Unix Kernel Statistics):hfxhum8	0.051	0.451
☑ 🟫	1	Collision Rate (Unix Kernel Statistics):ermdev1p	0	0

Source: http://bish.co.uk – Performance Testing – Page 17

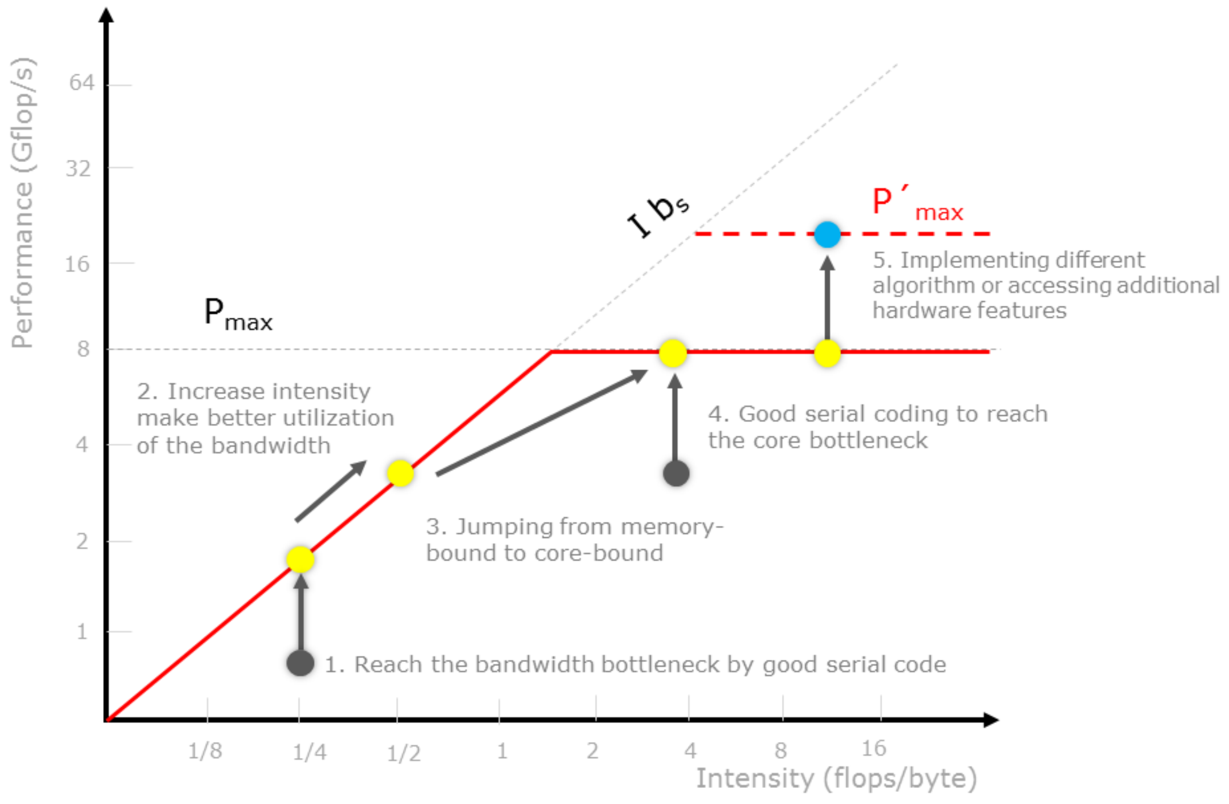
Figure 6.2: Service Analysis using HP LoadRunner

7. Summary

In the previous chapters, the concept of performance modeling was explained in details. Moreover the difference between the various performance models as well as the factors affecting it was clarified. Roofline model was discussed with an example; furthermore a refinement to the roofline model was explained in the fifth chapter which predicts the behavior of saturation and scaling of the bandwidth-limit. Furthermore, as mentioned in chapter five, how some HPC kernels have a constant arithmetic intensity and how it is connected with the local available store space. The sixth chapter helped us to understand the relation between the software optimization and the system performance. Additionally, the application and/or service modeling was introduced where some commercial as well as open-Source tools were suggested.

Although, the roofline performance modeling provides a visual assistance through a realistic forecast of performance and productivity of the system as well as shows hardware constraint for a given kernel. It is not suitable for those who are interested in fine tuning (+5%) as well as those who are challenged by parallel kernel correctness. Execution-Cache-Memory (ECM) describes the scaling characteristics of bandwidth bound codes on a multicore chip better than a simple bottleneck analysis. [15] [21] [22]

A recommendation optimization guide is shown in figure 7.1 where five steps will help to significantly improve the system performance. Starting from reaching the bandwidth bottleneck through a good serial coding then increasing the intensity by making a better utilization of bandwidth before jumping from memory-bound to core-bound then implementing a good algorithm.


Performance (Gflop/s)
64
32
16
8
4
2
1
I bs
P´max
Pmax
5. Implementing different algorithm or accessing additional hardware features
2. Increase intensity make better utilization of the bandwidth
4. Good serial coding to reach the core bottleneck
3. Jumping from memory-bound to core-bound
1. Reach the bandwidth bottleneck by good serial code
1/8
1/4
1/2
1
2
4
8
16
Intensity (flops/byte)

Source: http://moodle.rrze.uni-erlangen.de/pluginfile.php/10814/mod_resource/content/2/05_Roofline_Model.pdf, page 20

Figure 7.1: Recommendation to optimize system performance

Personally, I am convinced that, using roofline modeling plays a vital role in a big and a long-term projects where a lot of time, money and power are going to be invested and a small optimization will show a significant cost reduction as well as time and money saving.

References

[1] National Center for Supercomputing Applications (NCSA), University of Illinois, USA, "Performance Modeling for Systematic Performance Tuning", ACM 978-1-4503-0771-0/11/11, pp. 1 - 5.

[2] S. Jarvis, S.Wright, Simon Hammond, High Performance Computing Systems: Performance Modeling, Benchmarking and Simulation, Springer, ISBN: 978-3-319-10213-9, pp. 19 - 26.

[3] Heike McCraw, Innovative Computing Laboratory, Departmanet of Electrical Engineering and Computer Science, "Performance Modeling", University of Tennessee, 2013, pp. 7 - 9.

[4] Samuel Williams, David Patterson, ParLab Summer Retreat, "The Roofline Model: A pedagogical tool for program analysis and optimization", Berkeley Par Lab

[5] Samuel Williams, Andrew Waterman, David Patterson, "Simple performance modeling: The Roofline Model" Communications of the ACM, Vol. 52 No. 4, 10.1145/1498765.1498785, pp. 65 - 76.

[6] W. Schönauer (2000), "Scientific Supercomputing: Architecture and Use of Shared and Distributed Memory Parallel Computers"

[7] S. Williams (2008), "Auto-tuning Performance on Multicore Computers", UCB Technical Report No. UCB/EECS-2008-164. PhD thesis

[8] Datasheet Addendum, 2nd Generation Intel® Core™ Processor Family Mobile with ECC, May 2012

[9] Intel free Press, "Ron Friedman: The Man Behind Sandy Bridge", December 28, 2010. Retrieved November 11, 2011, URL: http://www.intelfreepress.com/news/the-man-behind-sandy-bridge/

[10] Samuel Williams, Lawrence Berkeley National Laboratory, "The Roofline Model", , Berkeley Par Lab

[11] Treibig J, Hager G. Introducing a performance model for bandwidth-limited loop kernels. Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science, vol. 6067, Wyrzykowski R, Dongarra J, Karczewski K, Wasniewski J (eds.). Springer Berlin / Heidelberg, 2010; 615–624, doi:10.1007/978-3-642-14390-8 64.

[12] G. Hager, J. Treibig, J. Habich, and G. Wellein, Erlangen Regional Computing Center (RRZE), "Exploring performance and power properties of modern multicore chips via simple machine models", Erlangen, Germany, pp. 1 - 21.

[13] Rotem E, Naveh A, Ananthakrishnan A, Rajwan D, Weissmann E. Power-management architecture of the Intel microarchitecture code-named Sandy Bridge. IEEE Micro 2012; 32:20–27, doi:10.1109/MM.2012.12.

[14] HP LoadRunner User Manual, v11.00, URL: http://community.hpe.com/hpeb/attachments/hpeb/sws-LoadRunner_SF/11042/1/hp_man_LoadRunner11.00_AnalysisUser_pdf.pdf

[15] Performance Modeling: The Roofline Model, "Loop-based performance modeling: Execution vs. data transfer", Friedrich-Alexander Universität, Erlangen-Nürnberg, pp 2-16

[16] Schönauer W. Scientific Supercomputing: Architecture and Use of Shared and Distributed

Memory Parallel Computers. Self-edition, 2000. URL http://www.rz.uni-karlsruhe.de/~rx03/book.

[17] Treibig J, Hager G. Introducing a performance model for bandwidth-limited loop kernels. Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science, vol. 6067, Wyrzykowski R, Dongarra J, Karczewski K, Wasniewski J (eds.). Springer Berlin / Heidelberg, 2010; 615–624, doi: 10.1007/978-3-642-14390-8 64.

[18] Suleman MA, Qureshi MK, Patt YN. Feedback-driven threading: power-efficient and high performance execution of multi-threaded workloads on CMPs. SIGARCH Comput. Archit. News Mar 2008; 36(1):277– 286, doi:10.1145/1353534.1346317.

[19] Hoisie A, Lubeck O, Wasserman HJ. Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional wavefront applications. Int. J. High Perform. Comp. Appl. 2000; 14:330–346, doi:10.1177/109434200001400405.

[20] Nudd GR, Kerbyson DJ, Papaefstathiou E, Perry SC, Harper JS, Wilcox DV. Pace — A toolset for the performance prediction of parallel and distributed systems. Int. J. High Perform. Comp. Appl. 2000; 14(3):228–251, doi:10.1177/109434200001400306.

[21] Kerbyson DJ, Jones PW. A performance model of the Parallel Ocean Program. Int. J. High Perform. Comp. Appl. 2005; 19:261–276, doi:10.1177/1094342005056114.

[22] G. Hager, J. Treibig, J. Habich, and G. Wellein, "Exploring performance and power properties of modern multicore chips via simple machine models", Erlangen Regional Computing Center (RRZE) Martensstr. 1, 91058 Erlangen, Germany