Analysis and modeling of

Computational Performance

Lecture 11

Roofline performance model

Execution time modelling

- Execution time modelling can be done for different codes, taking into account particular characteristics of the programs and application domains
  - in the following we will consider algorithms for which the execution time depends mainly on:
    - T_{calc} – the time for performing arithmetic operations (calculations)
    - T_{mem} – the time for memory accesses
- In the case of subsequent execution of separate operations, the execution time T_{exec} for a given algorithm can be obtained as a sum of non-overlapping parts, e.g.:
    - $T_{exec} = T_{calc} + T_{mem}$
- In practical computations, calculations and memory accesses are often done concurrently so the respective times can overlap
  - in the case of full overlap, execution time can be estimated as
    - $T_{exec} > \max(T_{calc}, T_{mem})$

Execution time modelling

- It is possible for a given program to **estimate** separately **limits** for T_{calc} and T_{mem}
  - assuming the code performs N_o operations and N_m memory accesses:
  - T_{calc} cannot be shorter than the time for performing arithmetic operations of the algorithm with the maximal performance offered by the hardware platform
    - $T_{calc} > N_o / P_o^{max}$ – where the maximal platform performance for operations, P_o^{max}, is expressed in GFLOP/s
  - T_{mem} cannot be shorter than the time for accessing data with the maximal performance offered by the hardware platform
    - $T_{mem} > N_m / P_{ma}^{max}$ – where the maximal platform performance for memory accesses, P_{ma}^{max}, is expressed in accesses/s (= 1/access_time)
    - the maximal platform performance for memory accesses can also be expressed in GB/s and denoted by P_{mB}^{max}
    - then: $T_{mem} > (N_m * size\_of\_data) / P_{mB}^{max} = N_{mB} / P_{mB}^{max}$

Execution time modelling

- The analysis can be further extended to take into account cache memories:
  - assuming the code transfers N_{cB} bytes from a given cache memory
  - given the maximal transfer rate P_{cB}^{max}
  - the transfer time T_{cache} cannot be shorter than N_{cB} / P_{cB}^{max}
- Finally the execution time can be estimated as longer than any of the separate limiting times:
  - for arithmetic (floating point) operations N_o / P_o^{max}
  - for DRAM memory accesses N_{mB} / P_{mB}^{max}
  - for cache accesses N_{cB} / P_{cB}^{max} (estimated for each cache level)
  - $\mathbf{T_{exec} > max(N_o / P_o^{max}, N_{mB} / P_{mB}^{max}, N_{c1B} / P_{c1B}^{max}, N_{c2B} / P_{c2B}^{max}, ...)}$
- Given the relation: $T_{exec} = N_o / P$, where P is the performance in GFLOPS
  - one can obtain the limit for the performance P given by:
    - $P = N_o / T_{exec} < N_o / \max(N_o / P_o^{max}, N_{mB} / P_{mB}^{max}, N_{c1B} / P_{c1B}^{max}, N_{c2B} / P_{c2B}^{max}, ...)$

Execution time modelling

- Performance modelling using execution time limits can be used to assess the actual execution time and possible optimizations
  - which of the limiting times is the longest?
  - can we decrease this times
    - for the limiting time used for arithmetic operations not much can be gained – the number of floating point operations is usually determined by the performed algorithm
    - for the limiting times related to memory and cache accesses, quite often it is possible to decrease the amount of data transferred, e.g. by some code reorganisation or modifications to data structures
  - how far the actual execution time is from the limiting times i.e. the limits imposed by the maximal performances (i.e. hardware)?
  - is their any room for improvement?
  - in which direction the further optimizations should go?
    - diminishing the amount of cache and memory transfers?
    - optimizing pipeline processing and cache and memory transfers?

Execution time modelling

- As an alternative to execution time considerations, another form of performance modelling, based on maximal performances possible to obtain for a given hardware, can be used
- The analysis, most often, is performed only for arithmetic operations and DRAM memory accesses
- The goal is to estimate the maximal performance in Gflop/s possible to obtain for a given code and a given hardware and to represent it graphically
- Given the graphic representation of maximal possible performances, the actual performance can be represented on the graph and the same questions posed:
  - how far the actual performance is from the limiting performance i.e. the limit imposed by the hardware?
  - is their any room for improvement?
  - in which direction the further optimizations should go?

Execution time modelling

- In order to create unified graphical representation of maximal performance possible to obtain on a given hardware, the notion of arithmetic intensity is introduced
  - for a given algorithm, its arithmetic intensity is the ratio of the number of operations to the number of memory accesses
    - $s_{pma} = N_o / N_m$ [flop/access]
      - **$s_{pmB} = N_o / (N_m * size\_of\_data)$ [flop/B]**
  - given arithmetic intensity, execution time can be estimated as
    - $T_{exec} > \max(N_o / P_o^{max} , N_o / (s_{pma} * P_{ma}^{max}))$
  - moreover, using the equation for the actual performance

    $P = N_o / T_{exec} < N_o / \max(N_o / P_o^{max}, N_o / (s_{pma} * P_{ma}^{max})$

    one finally arrives at the expression limiting the actual performance

    $P < \min(P_o^{max} , s_{pmB} * P_{mB}^{max})$

Execution time modelling

- One can arrive at the expression limiting the actual performance of given computations directly:
  - the performance must be lower than the maximal performance of the hardware:

    $\mathbf{P < P_o^{max}}$
  - moreover the performance, expressed in the number of operations performed in a given time, in the case where the execution pipelines can process only the number of operations for which the data have been transferred from the memory, can be limited by:
    - **P =** number_of_operations/execution_time

      = number_of_operations/number_of_transferred_data_items * number_of_transferred_data_items/execution_time

      = s_{pma} * performance_of_memory_transfers

      < s_{pma} * maximal_performance_of_memory_transfers
  - taking into account that the actual performance is lower than any of the limits (again assuming that operations and accesses are performed concurrently) leads to the final formula for limiting the actual performance:

    $$\mathbf{P < min(P_o^{max} , s_{pma} * P_{ma}^{max}) = min(P_o^{max} , s_{pmB} * P_{mB}^{max})}$$

Execution time modelling

- The expression

  $\mathbf{P < min(P_o^{max} , s_{pmB} * P_{mB}^{max})}$

  is the basis for the so called roofline performance model
  - the actual performance of the code on a given hardware is bounded by the performance obtained from the roofline model $\mathbf{P_r}$
    - $\mathbf{P < P_r = min(P_o^{max} , s_{pmB} * P_{mB}^{max})}$ (less often $=min(P_o^{max}, s_{pma} * P_{ma}^{max})$)
  - the diagram presenting $\mathbf{P_r}$ as the function of $\mathbf{s_{pmB}}$ (less often s_{pma}) is the so called roofline diagram
    - the diagram for a given hardware platform consist of two lines:
      - $\mathbf{P_o^{max}}$ - a horizontal line corresponding to the maximal performance of processor floating point pipelines
      - $\mathbf{s_{pmB} * P_{mB}^{max}}$ - a sloping line corresponding to the maximal DRAM memory throughput
      - the limiting performance can be found for any code, given its arithmetic intensity $\mathbf{s_{pmB}}$ (less often s_{pma})

Execution time modelling

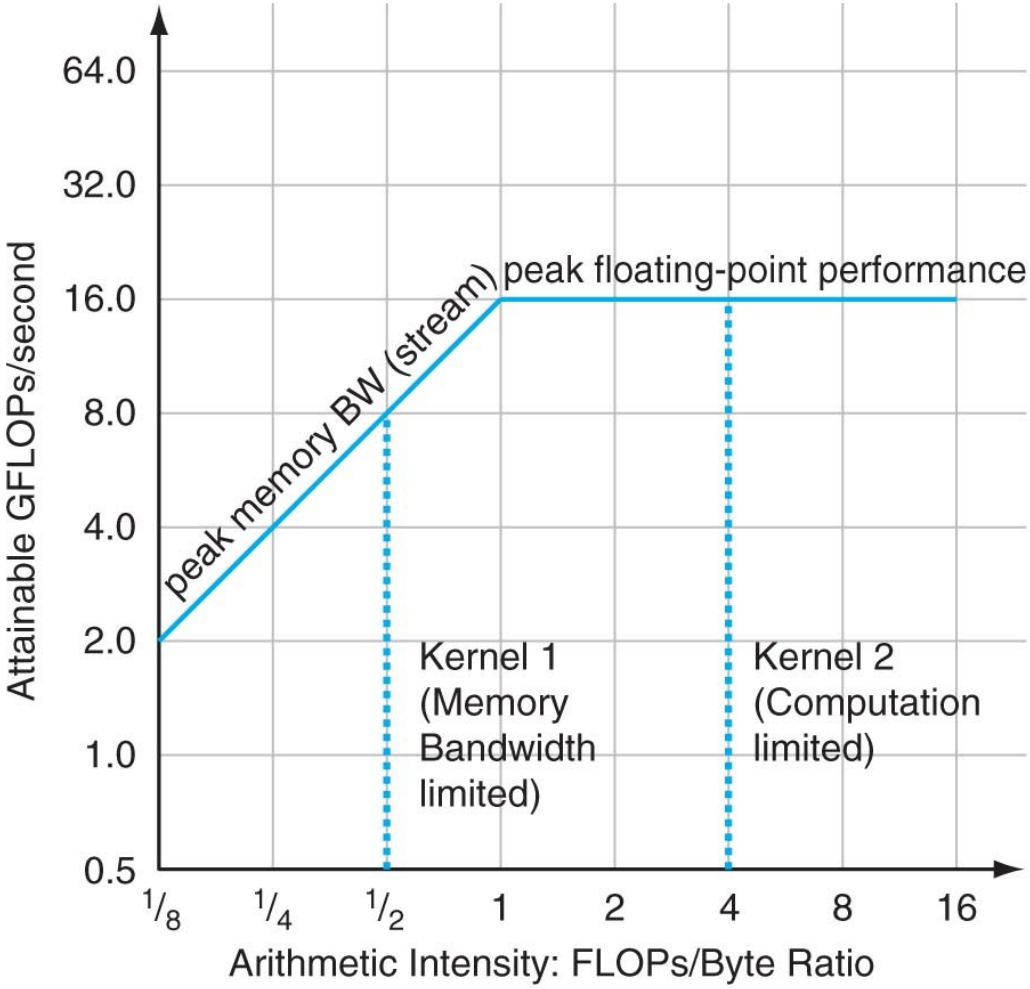
- The meaning of the expression

  $$P < \min(P_o^{max} , s_{pmB} * P_{mB}^{max})$$

  and the roofline diagram is that the performance possible to obtain for a given code on a given hardware platform depends on the value of arithmetic intensity
  - for the programs with low arithmetic intensity, their **performance is memory bound**
    - the processor cannot use its full processing power of pipelines, the pipelines wait for data from memory
      - the small number of arithmetic operations per data element (determined by s_{pm}) is done immediately, concurrently with the transfers of data for next operations - the performance is determined only by the rate at which data arrives to processor
  - for the programs with high arithmetic intensity, their **performance is processor bound**
    - the processor uses its full processing power, the data transferred (concurrently with computations) from memory is always ready

Roofline performance model

- The diagram of roofline performance model for an example platform
  - peak performances are obtained either from hardware characteristics or from micro-benchmarks
  - algorithms (kernels) are characterized only by its arithmetic intensity
  - the diagram shows the maximum available performance on the platform for a kernel with a given arithmetic intensity

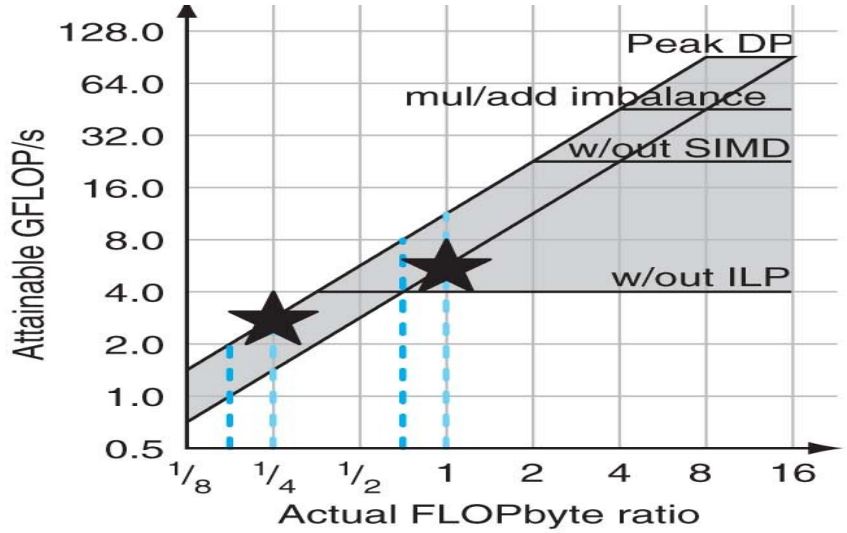


- The, so called, "ridge point" on the roofline diagram is an important platform characteristics showing the, so called, machine balance – the limiting arithmetic intensity beyond which the maximum processing performance can be obtained

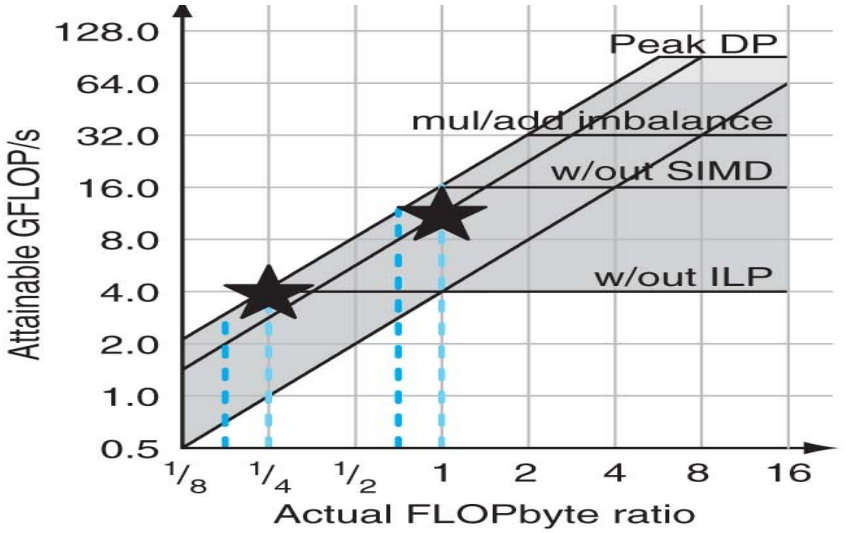
Roofline performance model

- The simple roofline performance model can be extended or modified in many ways:
  - theoretical or experimental (benchmark) performances can be used for drawing limiting lines
  - additional lines can be added to show the available platform potential in different situations
  - for processing power it can include lines for
    - utilization of vector (SIMD) capabilities, proper utilization of pipeline processing (ILP – instruction level parallelism), utilization of special hardware capabilities (e.g. FMA – fused multiply-add)
  - for memory performance it can include
    - utilization of hardware prefetching, speculative execution, NUMA affinity
    - there are special extensions for the, so called, cache aware roofline model
  - the most difficult in the effective use of the roofline model is the estimation of program arithmetic intensity for complex codes with sophisticated data structures and memory access patterns

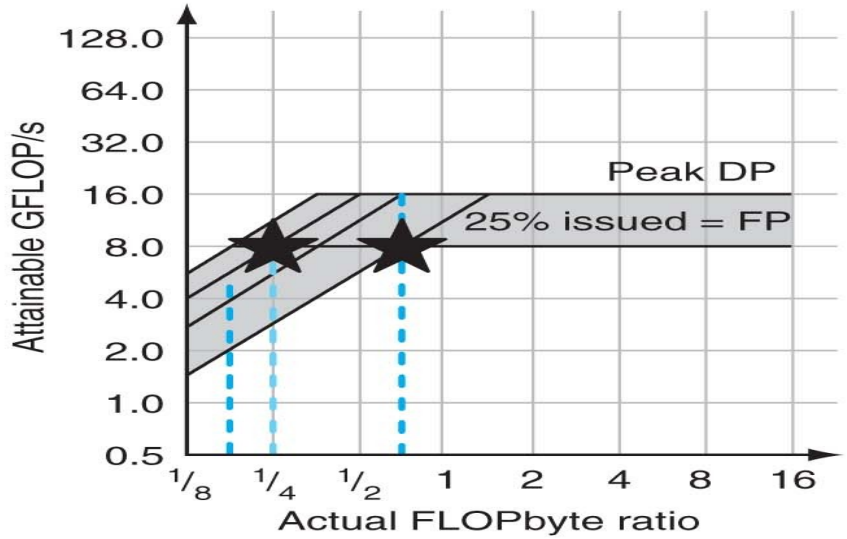
Roofline performance model



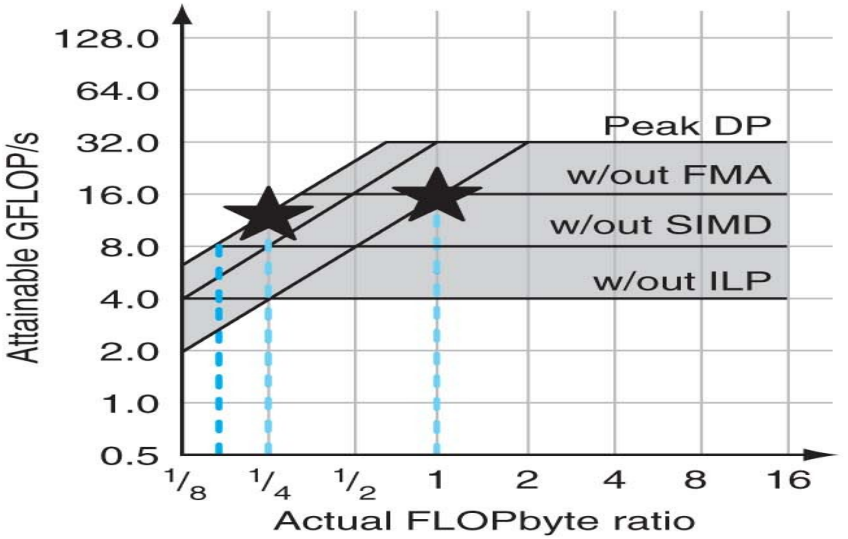
a. Intel Xeon e5345 (Clovertown)



b. AMD Opteron X4 2356 (Barcelona)



c. Sun UltraSPARC T2 5140 (Niagara 2)

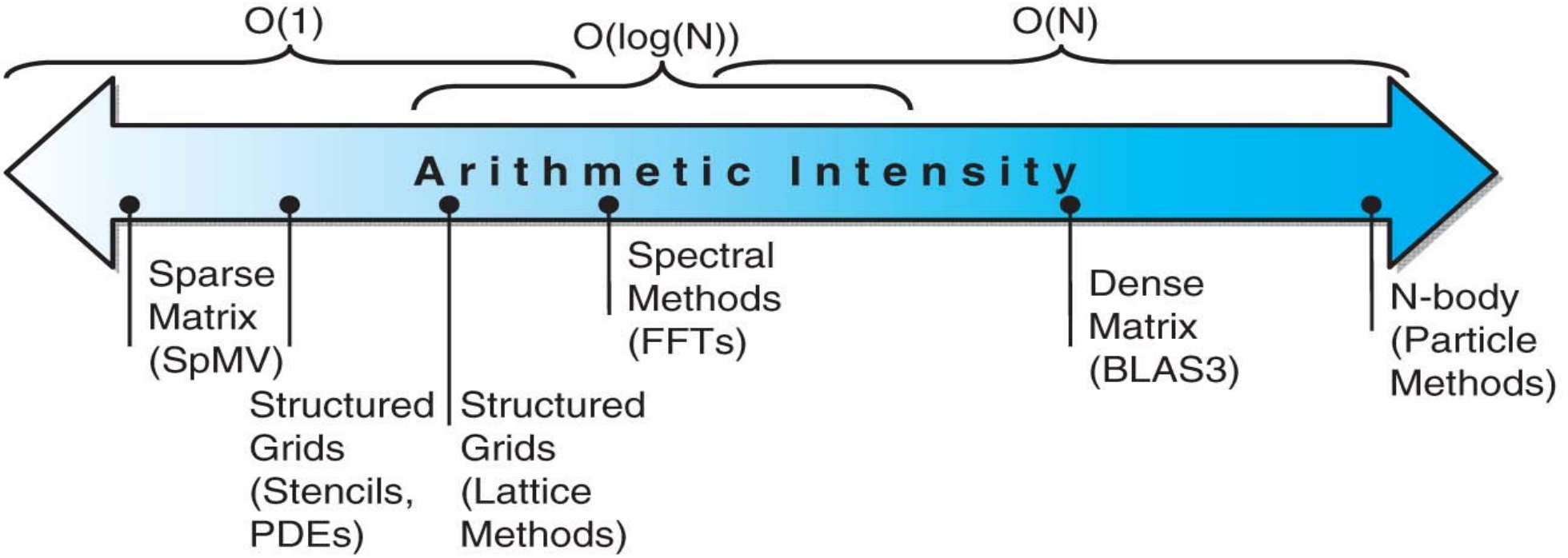


d. IBM Cell QS20

Roofline performance model

- How to obtain limiting lines for the roofline diagram of a given platform
  - separate diagrams can be constructed for single thread and multithreaded applications
  - for processing power in Gflop/s one can use:
    - theoretical estimates – number of operations per cycle for a single core * frequency in GHz [* number of cores]
    - results of microbenchmarks, such as e.g. matrix-matrix multiplication
    - results of performance tests, such as e.g. Linpack
  - for memory throughput
    - theoretical estimates – based on processor, bus and memory modules characterization
    - results of microbenchmarks – for different memory access patterns
    - results of performance tests, such as e.g. STREAM (which can be also classified as microbenchmark, due to its simplicity)

Arithmetic intensity



- Different kinds of algorithms are characterized by typical arithmetic intensities
  - given the dominating algorithm in the program, one can predict, based on the algorithm's arithmetic intensity, the actual performance of the program for a given platform and select platforms best suited for the program execution

Arithmetic intensity

- An example: matrix-vector product for dense matrices

```
for(i = 0; i < n, i++){
  for(j = 0; j < n; j++) {
    y[i] += a[i*n+j] * x[j];
} }
```

  - first obvious optimization (must be considered since should be done by any reasonable optimizing compiler)

```
for(i = 0; i < n, i++){
  temp = 0.0;
  for(j = 0; j < n; j++) {
    temp += a[i*n+j] * x[j];
  }
  y[i] = temp;
}
```

  - the number of DRAM memory accesses based on a simple analysis of the source code – 2 per iteration (+ *n* accesses to y)

Arithmetic intensity

- An example: matrix-vector product for dense matrices

```
for(i = 0; i < n, i++){
  for(j = 0; j < n; j++) {
    y[i] += a[i*n+j] * x[j];
} }
```

  - the number of operations N_o is assumed to be always the same
    - $N_o = 2*n*n$
  - the simple analysis based on the source code lead to the estimate of the number of bytes loaded from memory
    - $l = (2*n*n + n) * size\_of\_data$
  - the simple analysis does not take into account possible optimizations (manual or automatic, software or hardware)
    - one of optimizations might be the effective use of cache memory
    - in the ideal case of large enough cache the matrix `a` and the vector `x` are loaded from memory only once
    - hence the total number of bytes loaded from memory is
      - $l = (n*n + 2*n) * size\_of\_data$

Arithmetic intensity

- An example: matrix-vector product for dense matrices

```
for(i = 0; i < n, i++){
  for(j = 0; j < n; j++) {
    y[i] += a[i*n+j] * x[j];
} }
```

  - calculating s_{pmB} (for double precision computations)
    - original analysis
      - $s_{pmB} = 1 / (8 + O(1/n))$
    - theoretical ideal
      - $s_{pmB} = 1 / (4 + O(1/n))$
    - register blocking
      - $s_{pmB} = 1 / (6 + O(1/n))$

- register blocking optimization

```
for(i = 0; i < n, i+=2){
  ty0 = 0.0;
  ty1 = 0.0;
  for(j = 0; j < n; j+=2) {
    tx0 = x[j];
    tx1 = x[j+1];
    ty0 += a[i*n+j] * tx0;
    ty0 += a[i*n+j+1] * tx1;
    ty1 += a[(i+1)*n+j] * tx0;
    ty1 += a[(i+1)*n+j+1] * tx1;
  }
  y[i] = ty0;
  y[i+1] = ty1;
}
```

Arithmetic intensity

- Implementation horror – strided memory accesses to a

```
for(j = 0; j < n; j++) {
  for(i = 0; i < n, i++){
    y[i] += a[i*n+j] * x[j];
} }
```

  - for each iteration of inner loop an element of a is accessed, separated by *n* elements from the elements accessed in the previous and the next iterations, so new cache line must be loaded from memory for each iteration (no spatial and temporal locality)
  - when the next element in cache line is accessed, the line is no longer in any cache (for sufficiently large *n*)
  - the number of bytes loaded from memory is more than *n*n*size_of_cache_line* instead of less than *2*n*n*size_of_data*
  - the real (not effective) arithmetic intensity is
    - $s_{pmB} < 2$ / size_of_cache_line (usually $s_{pmB} < 2/64$ B)
  - the performance is horrible

Roofline performance model

- Roofline performance model can be used to estimate how far from the theoretical maximum the performance of a real program is
  - arithmetic intensity is used to position the code on the horizontal axis and the actual performance is put on the vertical line at that point
  - the distance from the actual performance to the limiting line shows how much can be possibly gained by optimization
    - the diagram allows for fast estimation of the ratio of the actual performance to the theoretical peak achieved by a given code
    - the possible optimizations of the code can include modifications that increase the arithmetic intensity, moving the code to the region with a higher performance bound
  - to reliably estimate the arithmetic intensity, the investigations of the real number of memory accesses to different levels of memory hierarchy should be done
    - one can use assembly code inspection, hardware counters and profilers that estimate the number of memory accesses (cache misses, etc.)