Performance Engineering
Assignment 2: Analytical modeling
Assigned: April 21st,
Due: Monday, May 9, 20:00.

Daniël Boelman, Dimitri Hooftman, Chaitanya Kumar, Kevin Kuurman

1 Introduction

The goal of this assignment is to demonstrate how to build and calibrate analytical models, for both sequential and parallel code. To this end, we use matrix multiplication and histogram as applications of interest.

This report describes the process of building and validating analytical models, as well as it reflects on the pros and cons of this type of performance models. Therefore, as background, we describe the general challenges of analytical models, and further discuss the specifics of modeling the matrix multiplication and histogram themselves, explaining the model building and calibration. We also include the resulting models and their validation. Finally, we summarize our findings and lessons learned from these two different analytical models.

2 Background

In this section we discuss the background knowledge necessary to build an analytical model.

An analytical model combines system and application knowledge to provide in-depth performance information. Specifically, the system is modeled in terms of ”operations” (and their cost), and the application is ”reformulated” using these operations. The typical challenges of building an analytical model are, therefore, the following:

- Define the abstract system operations and their cost.
- Express the application as a symbolic model based on ”abstract” system operations.

Define and calibrate the system operations

Q1 (0.5p): Explain how system operations can be defined.

System operations can be defined as operations useful for creating a model of the inner workings of an application. At a high level, these are for instance arithmetic and memory access operations. Depending

on the expected accuracy and the goal of the model, these operations can potentially be further divided into more specific operations. Increasing the granularity of the operations reduces the error not accounted for by a more abstract model. At the same time the model complexity will increase as it consists of more parameters being used in the calculation.

An example of dividable operation would for instance be a read from memory. Further dividing the read from memory by accounting for cache behaviour reduces the error in the models predictive power, but does add complexity as the cache behaviour has to be modelled.

Q2 (0.5p): Explain how the cost of system operations can be determined.

To determine the cost of a system operations, two approaches can be used. The first option is to determine the theoretical cost by using a catalogue of specific hardware. This catalogue can for instance provide the amount of cycles needed to perform an operation. The second option is to isolate an operation and microbenchmark its performance. Microbenchmarking tools can provide cycle information about an operations executed on specific hardware.

Symbolic application model

Q3 (0.5p): Explain how to define an analytical model for an application, and how a symbolic form is expressed.

To define an analytical model of an application, two parts are required. The first part is a model of the application or the code itself. This model only specifies relevant operations, this could mean that for instance looping, branching, or initialization operations are excluded. This model is build up from system operations. The second part is a model of the hardware. This model represents the cost of performing a system operations on theoretical hardware.

Combined, for a theoretical workload, these two models indicate the theoretical amount of operations that will be performed at a theoretical cost. The symbolic form of this analytical model defines the theoretical parameters by labels. These labels can then be used in combination with the hardware model by means of mathematical operations like addition, subtraction, multiplication and division representing the abstract working of an application.

Putting it all together

Q4 (0.5p): What are the differences between the symbolic and calibrated models? What can we learn from the different models?

The symbolic model of an application defines its theoretical working. This model can be used to, for instance, compare different applications. Calibrating the models fills in the hardware cost parameters corresponding to the system operations. These cost parameters are specific to the hardware being modelled and allow for comparison of an application being executed on different implementations of hardware. The calibrated model in combination with a defined workload can be used to predict the performance of the relevant functionality of an application.

3 Modeling matrix multiply (3p)

In this section we provide a detailed explanation of our modeling for matrix multiplication. The two versions we are using are: Q5: Please provide here a high-level pseudo-code of your matrix multiplication versions: sequential and parallel.

Algorithm 1 Matrix Multiplication Optimised Sequential Implementation

Input: matrices A[$m \times n$], B[$n \times p$] and C[$m \times p$]

1: $b = W/F$ ▷ Calculate tile width , Loop is in parallel
2: **for** i_b in $0 : b : (m-1)$ **do** ▷ Iterate through tiles of matrix using step size b
3: **for** j_b in $0 : b : (p-1)$ **do**
4: **for** k_b in $0 : b : (n-1)$ **do**
5: `microkernel`$(A[i_b, k_b], B[k_b, j_b], C[i_b, j_b])$ ▷ Perform algorithm 2 to tile
6: **end for**
7: **end for**
8: **end for**

Algorithm 2 The tile microkernel

Input: matrices A[$m_b \times n_b$], B[$n_b \times p_b$] and C[$m_b \times p_b$]

1: **for** i in $i_b..(i_b + b)$ **do**
2: **for** j in $j_b..(j_b + b)$ **do**
3: $d = 0$
4: **for** k in $k_b..(k_b + b)$ **do**
5: $d \mathrel{+}= A[i, k] \cdot B[k, j]$
6: **end for**
7: $C[i, j] = d$
8: **end for**
9: **end for**

Modeling and calibration

Q6: What is your symbolic model for the sequential matrix multiplication? Explain how you have built it. For both matrix multiplication we assume the matrix A to be of size $m \times n$, B of size $n \times p$ and C of $m \times p$. This gives our base analytical model of the sequential version to be:

$$T_{\text{SEQ}} = T_{\text{comp}} + T_{\text{mem}} \tag{1}$$

Where T_{comp} gives us the folowing equation:

$$T_{\text{comp}} = \frac{m}{b} * \frac{p}{b} * \frac{n}{b} * b * b * (T_{add} + b(T_{\text{mul}} + T_{\text{add}})) \tag{2}$$

This is because per tile we do $b * b * b$ add and mul operations for every entry of the row and column of the tile. We also do $b * b$ add operations as tiles can overlap thus we sum all solutions for one location of c. For

Algorithm 3 Matrix Multiplication OpenMP Parallel Implementation

Input: matrices A[$m \times n$], B[$n \times p$] and C[$m \times p$]

$b = W/F$ ▷ Calculate tile width
for i_b in $0 : b : (m-1)$ **do** ▷ This loop is in parallel
  for j_b in $0 : b : (p-1)$ **do** ▷ This loop is in parallel
    for k_b in $0 : b : (n-1)$ **do**
      `microkernel`$(A[i_b, k_b], B[k_b, j_b], C[i_b, j_b])$ ▷ Perform algorithm 2 to tile
    end for
  end for
end for

Algorithm 4 The tile microkerne (Unchanged between versions)l

Input: matrices A[$m_b \times n_b$], B[$n_b \times p_b$] and C[$m_b \times p_b$]

for i in $i_b..(i_b + b)$ **do**
  for j in $j_b..(j_b + b)$ **do**
    $d = 0$
    for k in $k_b..(k_b + b)$ **do**
      $d \mathrel{+}= A[i, k] \cdot B[k, j]$
    end for
    $C[i, j] = d$
  end for
end for

memory this gives:

$$T_{\text{mem}} = \frac{m}{b}\frac{p}{b}\frac{n}{b} \cdot b \cdot b \cdot (2 * T_{\text{read}} + 2T_{\text{write}} + b * (3T_{\text{read}} + 1T_{\text{write}}))$$

Which we get from the fact that for every solution in the $b \cdot b$ tile we read the value that tile entry already has and add it to the product of the row-column multiplication. This results in 2 read and 2 write operations. Furthermore to get the row-column multiplication product we need to do 3 reads and 1 write $b \cdot b \cdot b$ times. As we read the value in matrix A, matrix B and the current sum value and write that to the sum. Thus gives our final model:

$$\begin{aligned} T_{\text{seq}} &= \frac{m}{b}\frac{p}{b}\frac{n}{b} \cdot b \cdot b \cdot T_{\text{add}} + b(T_{\text{mul}} + T_{\text{add}})) + \frac{m}{b}\frac{p}{b}\frac{n}{b} \cdot b \cdot b \cdot (2T_{\text{read}} + 2T_{\text{write}} + b(3T_{\text{read}} + 1T_{\text{write}})) \\ &= \frac{m}{b}\frac{p}{b}\frac{n}{b} \cdot b \cdot b(T_{\text{add}} + 2T_{\text{read}} + 2T_{\text{write}} + b(3T_{\text{read}} + T_{\text{write}} + T_{\text{add}} + T_{\text{mul}})) \end{aligned} \tag{3}$$

Q7: Explain how you calibrated your sequential symbolic model for matrix multiplication. To calibrate this we use the Agner[1] report. We calibrate the model for the CPU on the DAS-5 nodes. The CPU is an Intel Xeon E5-2630-V3 8 core 16 thread CPU. It is from the Haswell family so we use the values benchmarked for the Haswell architecture from the report. Our tile size is 16. Our CPU has an average clock speed of 1.59GHz. This gives us a calibrated model in the form of:

$$\begin{aligned} T_{seq} &= \frac{m}{16}\frac{p}{16}\frac{n}{16} \cdot 16 \cdot 16 \cdot (1/1.59 + 2 \cdot 0.5/1.59 + 2 \cdot 0.5/1.59 + 16 \cdot (3 \cdot 0.5/1.59 + 0.5/1.59 + 1/1.59 + 2/1.59)) \\ &= \frac{m}{16}\frac{p}{16}\frac{n}{16} \cdot 13363,522 \end{aligned} \tag{4}$$

We can see that the total execution time is thus input dependent.

Q8: What is your symbolic model for the parallel matrix multiplication? Explain how you have built it, focusing on the differences from sequential matrix multiplication. The parallel base model is similar to the sequentional version. The only differences is that the computation time and memory access time is spread over the threads. We also have an add time as there is some communication which takes time aswell as some parallel overhead.

$$T_{\text{Par}} = (T_{\text{comp}} + T_{\text{mem}})/T_{\text{processors}} + T_{\text{comm}} + T_{\text{par}} \tag{5}$$

This gives us for the computation the following equation:

$$\begin{aligned} T_{\text{comp}} &= \frac{N_{\text{alu\_ops}} \cdot T_{\text{alu\_ops}}}{P_{\text{processors}}} \\ &= \frac{\frac{m}{b}\frac{p}{b}\frac{n}{b} \cdot b \cdot b(T_{\text{add}} + b(T_{\text{add}} + T_{\text{mul}}))}{P_{\text{processors}}} \end{aligned} \tag{6}$$

Which is the same as sequential just spread over the threads. For the memory access this would be:

$$\begin{aligned} T_{\text{mem}} &= N_{\text{mem ops}} \cdot T_{\text{mem op}}/P_{\text{processors}} \\ &= \frac{\frac{m}{b}\frac{p}{b}\frac{n}{b} \cdot b \cdot b(2T_{\text{read}} + 2T_{\text{write}} + (b(3T_{\text{read}} + T_{\text{write}})))}{P_{\text{processors}}} \end{aligned} \tag{7}$$

[1] https://www.agner.org/optimize/instruction_tables.pdf

Which again is the same as sequential spread over the threads.

The total communication time should be 0. This is because theoretically we only parallelise the outer 3 for-loops. We don't parallelise the tiles. This means that data synchronisation is not needed. As is inter thread communication or any thread locking operation. Thus:

$$T_{\text{comm}} = 0 \tag{8}$$

We will have some parallel operation. As we don't posses the tools or knowledge to break this down in a more accurate modeled equation it will just remain the constant overhead. Thus:

$$T_{\text{par}} = T_{\text{overhead}} \tag{9}$$

The our fully realized model would be:

$$T_{PAR} = \frac{\frac{m}{b}\frac{p}{b}\frac{n}{b}b \cdot b(2T_{\text{read}} + 2T_{\text{write}} + b(3T_{\text{read}} + T_{\text{write}})))}{P_{processors}} + \frac{\frac{m}{b}\frac{p}{b}\frac{n}{b}b \cdot b(T_{\text{add}} + b(T_{\text{add}} + T_{\text{mul}}))}{P_{\text{processors}}} + 0 + T_{\text{overhead}} \tag{10}$$

This means that the parallel version should be exponentially faster as the thread count increases compared to the sequential. Of course the overhead will increase with amount of threads but it will not increase linear or exponentially. Thus we still keep the constant. So in most cases the parallel version should be faster than the sequential

Q9: Explain how you calibrated your parallel symbolic model for matrix multiplication, again focusing on the differences from the sequential version. The calibrating step is pretty much the same as the sequential except for the loops unrolled over threads. Our calibrated model would be:

$$\begin{aligned}
T_{\text{PAR}} &= \frac{\frac{m}{b}\frac{p}{b}\frac{n}{b}b \cdot b(2T_{\text{read}} + 2T_{\text{write}} + b(3T_{\text{read}} + T_{\text{write}})))}{P_{\text{processors}}} + \frac{\frac{m}{b}\frac{p}{b}\frac{n}{b}b \cdot b(T_{\text{add}} + b(T_{\text{add}} + T_{\text{mul}}))}{P_{\text{processors}}} + T_{\text{comm}} + T_{\text{overhead}} \\
&= \frac{\frac{m}{16}\frac{p}{16}\frac{n}{16}16 \cdot 16(2 \cdot 0.5 \cdot 1.59 + 2 \cdot 0.5 \cdot 1.59 + 16(3 \cdot 0.5 \cdot 1.59 + 0.5 \cdot 1.59)))}{P_{\text{processors}}} \\
&+ \frac{\frac{m}{16}\frac{p}{16}\frac{n}{16}16 \cdot 16(1 \cdot 1.59 + 16 \cdot (1.59 \cdot 1 + 2 \cdot 1.59))}{P_{\text{processors}}} + T_{\text{comm}} + T_{\text{overhead}} \\
&= \frac{\frac{m}{16}\frac{p}{16}\frac{n}{16} \cdot 5474.213836}{P_{\text{processors}}} + \frac{\frac{m}{16}\frac{p}{16}\frac{n}{16}7889.308176}{P_{\text{processors}}} + T_{\text{comm}} + T_{\text{overhead}}
\end{aligned} \tag{11}$$

Which uses the same values as the sequential version

Validation

Q10: Explain how you validated your models. We validated the models by testing matrices of different sizes in a few different configurations. We also run both implementations 10 times to get an average times so we eliminate as much fluctuation as possible. We measure the time in nanoseconds as this is needed for our smallest matrices. For sequential we get this table: We can clearly see that our model is quite a bit off in

m	p	n	Pred. performance (nanosec)	Act. performance (nanosec)
5	5	5	407.8	1295.4
130	130	130	7167885.219	12425717.800000
958	292	958	874328941.9	877190660.800000
292	958	292	266496921.8	280260115.200000
1454	1454	1454	10028920900	9831473062.000000

Number of threads	m	p	n	Pred. performance (nanosec)	Act. performance (nanosec)
4	5	5	5	101.9555817565918	22615.200000
16	130	130	130	447992.82623846433	1672740.600000
16	958	292	958	54645558.9206	76053961.200000
16	292	958	292	16656057.6251	26616751.200000
16	1454	1454	1454	626807554.788	798020374.000000
230644371.600000					

the first two cases. Later this is not noticable. One of our explanations for this would be the fact that when the matrices are so small the tiling actually negatively increases peformance and creates a lot of overhead which has a lot of influence on the total runtime as it is so short. In the last three test cases we see that the model is pretty close with the measure values. This is because at these matrix size tiling works well and the overhead of the tiling is so small it is not really measurable. The reason why its still not equal would be the fact that our processor is never only at 1,59 Ghz throughout the whole process. Another thing is that small io and background processes could actually create some noise in the measured results.

We run our parallel with the same test cases as sequential otherwise any comparison is useless.

We can clearly see that, especially with the small matrices we never reach the predicted performance. This is because the overhead is such a big influence our model is not very close. This is because we can't accurately model overhead and thus it is not modeled correctly. Later we see our model is close but still is quite low compared to measured time. This is mostly because the overhead has a big influence and we don;t take it correctly in account. Q11: What is the accuracy of your models? Our models are not accurate. The sequential is quite close compared to the measured times. But not accurate is because we dont take the loop incrementers and matrix indexing in account as operations that take time. Our parallel model is more inaccurate. This is because we dont model the parallel overhead correctly. Thus our model is just a sequential version spread over threads. We also dont take the overhead of the matrix indexing and loop increment in a tile in account. This is our explanation for the difference between measured and predicted times.

4 Histogram (4p)

In this section we provide a detailed explanation of our analytical models for two parallel algorithms calculating the histogram. The two parallel versions we use are:

Q12: Please provide here a high-level pseudo-code of your histogram versions: Histogram1 and Histogram2.

Two parallel histogram kernels where modelled and implemented, both kernels where implemented using OpenMP. The first kernel detailed in algorithm 5 performs atomic operations in parallel. The second kernel detailed in algorithm 6 uses local bins to perform operations on in parallel. After the parallel phase, a sequential phase combines the local bins.

Algorithm 5 Histogram1

Input: data[n], bins[$bincount$] and $binsize$

```
1: for i in 0..n do                                  } in parallel
2:     atomic(bins[data[i/bin size]] += 1)           }
3: end for                                           }
```

Algorithm 6 Histogram2

Input: data[n], bins[bin_count] and bin_size

```
1: local_bins ← Vector(size: thread_count)
2: for i in 0..n do                                  } in parallel
3:     local_bins[thread_id][data[i/bin_size]] += 1  }
4: end for                                           }
5: for i in 0..bin_count do
6:     for j in 0..thread_count do
7:         bins[i] += local_bins[j][i]
8:     end for
9: end for
```

Modeling and calibration

Q13: What is your symbolic model for Histogram1? Explain how you have built it.

To build the symbolic model for algorithm 5, we start with a high level overview. The time the kernel takes is calculated by taking the time spend on all computations and memory operations and dividing them by the amount of threads on which the histogram kernel is run in parallel.

$$T_{h1} = N_{\text{input}} * (T_{\text{comp}} + T_{\text{mem}})/N_{\text{threads}} + T_{\text{comm}} \tag{12}$$

The time spent on communications contains the communication time and synchronization time needed to perform the kernel in parallel. An example of this is the time spend on the atomic operation in algorithm 5 as it uses a shared data structure which can not be used by multiple threads at the same time.

To create a more detailed analytical model the parameters are made more precise by modelling the specific computations, memory operations and communications. The computation performed for the kernel is an addition. The memory operations performed is a read of the value in the input, a read of the value in the bin and a write of a new value to the bin. The communications is taking into account the overhead spent on paralleling the kernel and is seen as a constant which is not taken into account for the calibration of the model.

$$T_{\text{comp}} = O_{\text{add}} \cdot T_{\text{add}} + O_{\text{div}} * T_{\text{div}} \tag{13}$$

$$T_{\text{mem}} = C_{\text{read}} * T_{\text{read}} + C_{\text{write}} * T_{\text{write}} \tag{14}$$

To take into account caching behaviour, the memory hierarchy is modelled with its expected hit ratio. For this analytical model only a L1 cache is assumed to be present. The hit time and miss time are filled while calibrating. The hit rate is assumed to be 90%.

$$T_{\text{read,write}} = \text{hit\_rate} * T_{\text{hit}} + (1 - \text{hit\_rate}) * T_{\text{miss}} \tag{15}$$

Q14: What is your symbolic model for Histogram2? Explain how you have built it, especially focusing on the differences (if any) compared to Histogram1.

The symbolic model for algorithm 6 follows the same structure for both computations and memory operations. The difference lies in the time needed for communications. This time is smaller because the atomic operations are not performed which means less overhead in locking and unlocking and also means less potential waiting time for other threads to access the data structure. The kernel of algorithm 6 does however add logic for local bins which adds parallelization time to the total execution adding an new parameter to the model.

$$T_{h2} = N_{\text{operations}} * (T_{\text{comp}} + T_{\text{mem}})/N_{\text{threads}} + T_{\text{comm}} + T_{\text{par}} \tag{16}$$

The sequential combination of local bins adds again the same computations and memory times used for filling the local bins. This time however this operation scales to the amount of bins and threads, which is presumably a relatively low number, making it unnecessary to execute this in parallel.

$$T_{\text{par}} = (T_{\text{comp}} + T_{\text{mem}}) * N_{\text{bins}} * N_{\text{threads}} \tag{17}$$

Q15: Explain how you calibrated your models, specifically focusing on the differences against matrix multiplication and the differences between Histogram1 and Histogram2.

For calibrating T_{h1}, the time for the atomic operations in micro-benchmarked because the time spend to perform an atomic operation is not modelled. To micro-benchmark the atomic operation on line 2 in algorithm 5, a high resolution timer was used providing a result of 85,28 nanoseconds(ns) per atomic operation. This time includes the read of the data, the division of the resulting data, the addition, the read and the write from and to the bins. The caching behaviour and locking overhead of the read and write is taken into account in the micro-benchmark result of the atomic operation. Filling in the parameters provides the following prediction for algorithm 5:

$$65.536.000 * 85,28ns/2 = 2.794.455.040ns \approx 2,8s \tag{18}$$

To calibrate for T_{h2} a different approach is used, because there is no atomic operation we fill in the parameters based on the performance reported in the instruction table of Agner[1] report.

[1] https://www.agner.org/optimize/instruction_tables.pdf

First the cycle time of the hardware is reported by the CPU to be 1,59 nanoseconds per cycle. The throughput of both the addition and division operations are used to calculated the time spend on T_{com}. The throughput of both the read and write operations are used in the same way to calculate the memory access time. The cache read hit time is decided to be 1 cycle of 1,59 nanoseconds, the cache read miss time is decided to be 50 cycles of 1,59 nanoseconds. The write times are divided by 2 as the throughput reported in the Agnes report is half of the write times.

To calculate the T_{mem} we also take into account the L1 cache choosing a 90% percent hit ratio for both the read and the write operations. In addition, time for combining the local bins is calculated. Filling in the parameters provides the following prediction for algorithm 6:

$$T_{\text{comp}} = 1,59ns + 17,49ns \tag{19}$$

$$T_{\text{mem}} = (2 * (0,9 * 1,59ns + (1 - 0,9) * 79,5ns)) + (0,9 * 0,795ns + (1 - 0,9) * 39,75ns) \tag{20}$$

$$65.536.000 * (T_{\text{comp}} + T_{\text{mem}})/2 + (T_{\text{comp}} + T_{\text{mem}}) * 26 * 2 = 1.393.706.218ns \approx 1.4s \tag{21}$$

Validation

Q16: Explain how you validated your models.

To validate the model, the goal of the model is defined. For this assignment the goal of the model is to determine which of the algorithms algorithm 5 and algorithm 6 performs faster to decide weather to use atomic operations or local bins for a histogram kernel. We would also like to see how each kernel is predicted to perform for different input sizes and threads counts. To validate the model, the expected performance is calculated for the different inputs. After this, a kernel implementation is run 5 times to get an average performance on the DAS5.

Model	Input (million)	Thread count	Pred. performance (sec)	Act. performance (sec)
Atomic	65,5	2	2,8	2,55
Local bins	65,5	2	1,4	1,29
Atomic	65,5	16	3,5	36,38
Local bins	65,5	16	1,74	2,77
Atomic	655	2	27,9	25,6
Local bins	655	2	13,93	10,82
Atomic	655	16	34,9	354
Local bins	655	16	17,4	23,2

Q17: What is the accuracy of your models?

Empirically the models seem useful for comparing the two kernels of the histogram algorithm showing that algorithm 6 is both predicted and measured to be faster than algorithm 5. Excluding both atomic model

predictions for 16 threads, the measured times are on average 23% off the predicted times. The predictions made by the atomic model for 16 threads are both around 900% of the measured value, this is probably due to the micro-benchmark value used for prediction. The micro-benchmark value is measured with two threads, the results for 16 threads could be different due to atomic overhead not being captured by the value currently used as parameter for the model.

5 Conclusion and Future Work

This section presents the lessons learned from this assignment. We specifically focus on the construction, calibration, and accuracy of analytical models.

Lessons learned

Q18 (0.5p): What are the challenges you encountered while building (all) the models in this assignment? How do they compare to the expected challenges?

Modelling the atomic operations being used in algorithm 5 was detailed and complex, making the model not generalize to different machine architectures. The modelling of these operations was replacing with a micro-benchmark of the operations on the DAS5. It seems like the amount of threads being used to execute the kernel has a large impact on the values retrieved from the micro-benchmark. To get a more accurate micro-benchmark of the atomic operations, the values should be measured for different amounts of threads. The downside of this approach is that the model gets more machine specific and loses some of its predictive power for the specific amounts of threads.

One of the hardest thing to model, which we didn't get working, is parallel overhead. This is one of the more challenging parts.

Q19 (0.5p): What do you think about the accuracy of your models, and their usability? Can these statements be generalized for more/all analytical models?

The histogram models seem usable to make a comparison between the two kernels. For its predictive power, it depends on the the required accuracy of the prediction. The atomic model does not seem accurate enough for multiple amounts of threads, making it useless when accurate predictions for multiple amounts of threads are expected. The benchmark results are also only tested on the DAS5, introducing a risk for the measurements to be machine specific.

We think that for the matrix multiplication the sequential model is accurate enough to use for predictions and comparison between kernels. The parallel version is too inaccurate to use for this.

Future work

Q20 (0.5p): Given more time, is there anything you would do to improve your models? What, how, why?

If possible, replace the micro-benchmark parameter in the atomic model by a symbolic model which generalizes to different architectures and different amounts of threads being used. The accuracy of the histogram

models could also be improved by allowing for more detailed cache behaviour. This detailed cache behaviour should also be calibrated using benchmark data to increase the accuracy. Another thing we would want to look into is correctly measuring the overhead in the parallel versions and correctly implementing this in our models. This is because it is important for a more accurate model to take this parameter in account and we currently do not. We think this will result in better prediction of total execution time. Another think that we would do is make a better test suite for matrix multiplication. We used the matrices that were given in assignment 1. If we had the time we would like to make matrices that have similar increments in size so we can get a better idea how input size influences matrix multiplication. This would also make it possible to spot potential patterns faster. Another avenue of research would be optimising the number of threads used in OpenMP. As currently we hard code it to 16 (except for 5x5 matrices). This is not always the most optimal amount of threads. If we took the time to optimize this it would be more interesting for potential results.