Design Study Report

Participants

Reece Karge (rkarge3)
Alexandr Dorofeyev (adorofeyev3)
Kimanii Daniel (kdaniel38)
Andre Pollard (apollard6)
Michael Tidwell (mtidwell3)

20 September 2014

Context

In this design study, multiple software solutions were compared in order to best solve the problem of simulating the diffusion of heat through a two (2) dimensional square metal plate. These programs use the same environment and inputs to see the tradeoffs between performance and precision. The result of this design study is to find the best balance between user friendliness, performance, precision, and accuracy.

Research Questions

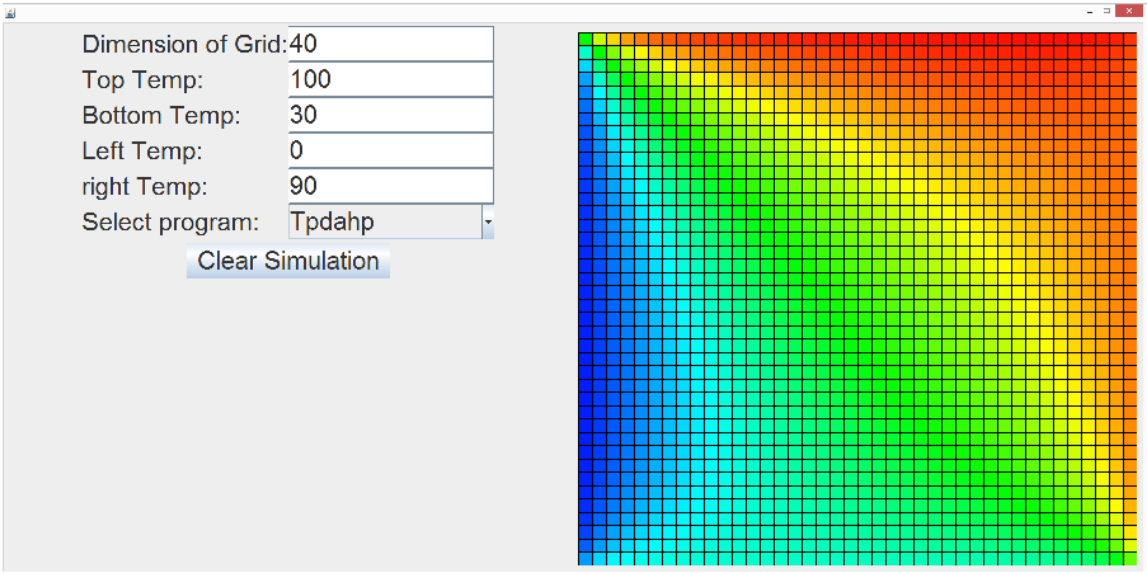
1. How is the performance affected by the change in grid size?
2. How is the performance affected with the change in precision?
3. How does usability affect lines of code in the GUI?
4. How does the lattice design structure affect the performance of the simulation?

Subjects

Subjects of this study are five variants of a simulation program:

1. Tpdahp - performed in double precision using a two-dimensional array of doubles to represent the plate
2. Tpfahp - using floating point computations on an array of floats
3. Twfahp - using floating point computations on an array of Floats
4. Tpdohp - The algorithm without using arrays. Instead, each lattice point should be represented using an object. Use doubles to hold temperature values.
5. Gallhp - A GUI version that enables the user to execute any of the four previous versions and to see a visualization of the results

Figure 1: GUI Layout



Experimental Conditions

The experiment has been conducted on a single computer with the following specifications:

Computer model	Macbook Pro Model A1286
Operating system	OS X Version 10.9.5
CPU	2.3 GHz Intel Core i7
Memory	8 GB 1333 MHz DDR3

Java	JRE 7

No virtual machine was used to conduct the experiments. No other programs were running while conducting the experiments. The programs were manually executed at the command prompt using the following command syntax:

"`java <packageName>.Demo -d # -l # -r # -t # -b #`"

This will be repeated for every program replacing "`<packageName>`" with Tpdahp or Tpfahp or Twfahp or Tpdohp, depending on the simulation and -d for the grid size, -l for the left, -r for the right, -t for the top and -b for the bottom temperatures.

Independent Variables

Variable	Units of Measurement	Description
GridSize	Numeric	The grid size is specified as an input parameter and it repres the size of the grid used in the simulation. the -g parameter
TopTemp	Degrees Celsius	This represents the top temperature, specified by the -t parame on the programs
BottomTemp	Degrees Celsius	This represents the bottom temperature, specified by the parameter
LeftTemp	Degrees Celsius	This represents the left temperature, specified by the -l parame
RightTemp	Degrees Celsius	This represents the right temperature, specified by the parameter
Precision	Number of significant digits	How many decimal points are kept through the calculations
RelativeChangeStoppingCriterion	Degrees Celsius	Minimum amount of relative change (from 0.0 to 1.0) for computation to continue
Model representation		How the heated plate is modeled a. Object oriented vs array b. Wrapped vs primitive

Dependent Variables

Variable	Units of Measurement	Description
MemoryUsage	Bytes	is expected to be affected by gridSize & precision
Lines of code	Numeric	is affected by model representation & precision
Time of execution	Milliseconds	is expected to be affected by gridSize & precision
Number of iterations	Numeric	steps taken until the result stabilize

Usability (for GUI only)

We studied the components used in our GUI design and how it affects the user experience as they conduct the experiment. We decided that an adequate study of usability was to create three variations of the GUI and note the time it took for new users to run the experiment and report the results on each design variation. Variations of the UI that we considered were as follow:

a. Changing the temperature text boxes to sliders or dials, where the user could slide a pointer left or right to decrease or increase the temperature value. This offers an easier control logic and also contains the user to sane inputs
b. Use grayscale for the color variations on the heating plate animation. Grey scale offers a better depth in perception of change.
c. Change the drop down list to option radio buttons so the user can easily see all the available options and reducing the amount of clicks to completion.

Adaptability

We used design patterns that were highly adaptable to changes in usages and implementation scenarios. We analysed various change scenarios and how resilient our design would be to these scenarios. We first looked at adding more simulators which begged the question

“How would the design be susceptible to more simulators?”. Simulators in this case refers to the various programs. Fortunately, we used the Template pattern, so adding a new simulator would be a matter of extending the DiffusionSimulator class and building out the exact implementation of the methods, everything else would fall into place.

Another adaptability scenario we considered was the ability to execute the experiment on different operating systems. This would require no change in the underlying code or design and the user experience would be mained across OSes because of the JVM infrastructure. The code runs in a Java Virtual Machine that will abstract the OS specific calls away from the code, so because of this we conclude the the code is very adaptable to change in operating systems

Yet another change scenario we thought about was the ability for this experiment to be conducted on a large scale. Imagine this experiment were to be expanded to a larger simulation, maybe grid size of upwards of 10,000. Currently the experiment is so intense that a grid size of over 1000 would stress any modern computer to a crawl. To be scalable the structure of the code will have to change dramatically, maybe introducing multi-processor, multithreading capabilities even to the extent of multi-machine (server farm). The current design is not scalable to higher simulation sizes.

Another change scenario would be to run all experiments serially then export the results of each, maybe even display computed statistics. This would require building a wrapper program that executed the current programs and also minor changes to the current programs as to how the results are returned to the wrapper program.

Metrics

These are the units of measurements used in the experiment.

Metric	Description

MilliSeconds	One thousandths of a second, which is a standard unit of measurement fo time. This will be measured using the internal java Date object
Bytes	A unit of memory size, containing 256 bits of data. This will be measure using the Runtime object
Degrees Celsius	Standard unit of measurement for temperature.
Lines of code	Count of distinct lines of codes (non-white space)
Number of significant digits	Digits that carry meaning, contributing to a number's precision.

Methods

Experiments were conducted on a single computer. The programs were run repeatedly with varying arguments, and results were recorded. Initial temperature values of top, left, right, and bottom edges of the plate were chosen arbitrarily as 1, 33, 66, 100 degrees respectively and did not change throughout the experiments.

For the first part of experiment the size of the plate was varied between the trials with the following values: 5, 10, 20, 50, 100, 200, 500, 1000. Four different programs (Tpdahp, Tpdohp, Tpfahp, Twfahp) were tested varying the GridSize, resulting in 4*8 = 32 trials. Each experiment was run three times, recording average value of resulting parameters between each run.

For the second part of the experiments the size of the plate was set to constant value equals to 100. Only one variant of program was run, which is Tpdahp. This time the relative change stopping criterion was changed taking the following values: 1, 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001, 0.00000001, 0.000000001, 0.0. After that, one program Tfdahp with float instead of double type of variable was executed one time with the change stopping criterion set to 0.0. Resulting time of execution, memory used and number of iterations values were recorded.

The Java Runtime class was used to measure the memory in bytes consumed by programs. The Java System class was used to measure time of execution in milliseconds. Number of iterations is counted in an integer variable while program is executed and outputted when execution is completed.

Analysis Techniques

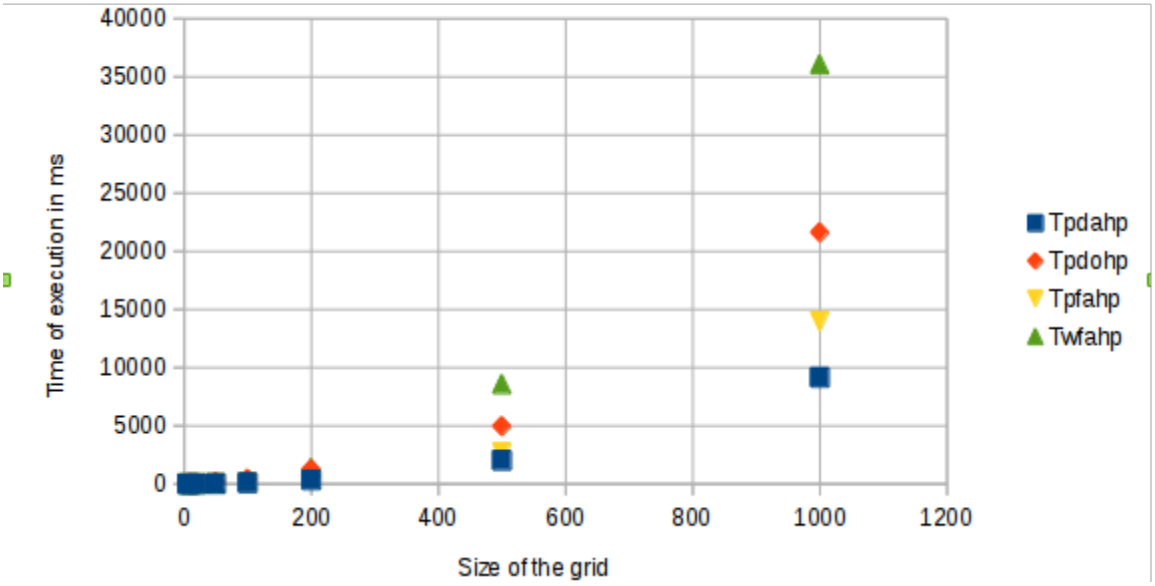
To visually analyse the data three charts were constructed. Correlation coefficient was computed for each chart using the online version of the Pearson Correlation Coefficient calculator[1].

Significance was originally represented as the relative change stopping criterion in the experiment. However after some deliberation, it has been decided to transfer these values to number of significant digits. It was assumed that relative change stopping criterion value 1 corresponds 0 significant digits, 0.1 corresponds to 1, 0.01 corresponds to 2, etc. Also, it was assumed that the change stopping criterion value of 0 corresponds to 17 significant digits, because in "double" type of Java variable 52 bits are used for mantissa, which results in approximately 17 significant digits.

Results

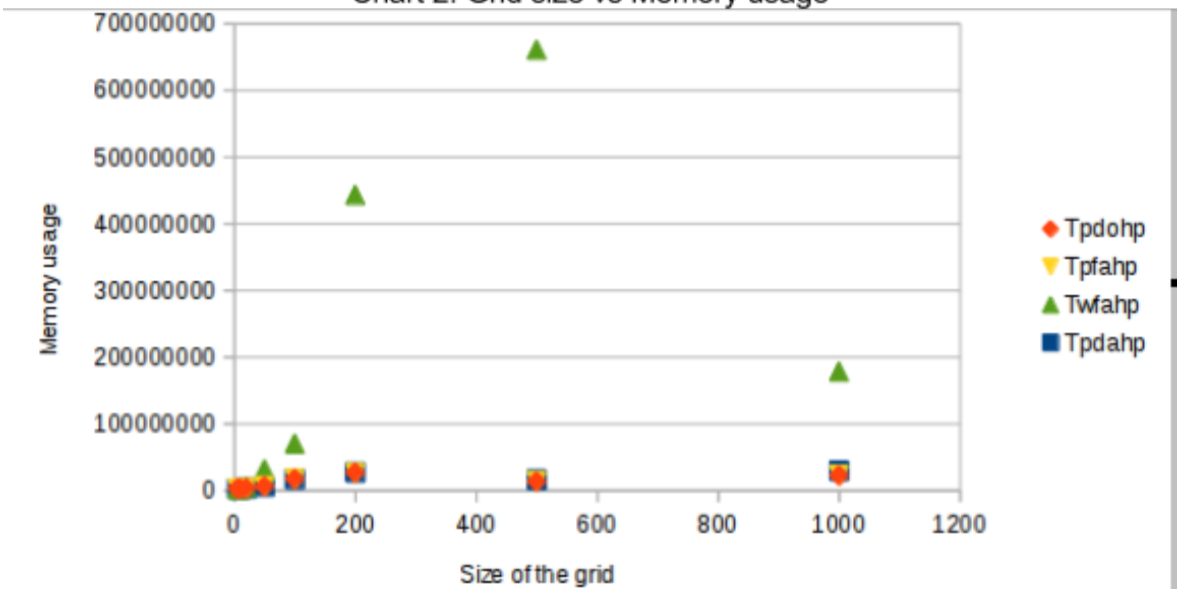
Results are shown in the charts below. Correlation coefficient R = 0.96 for all programs.

Chart 1: Size of Grid vs Time of execution



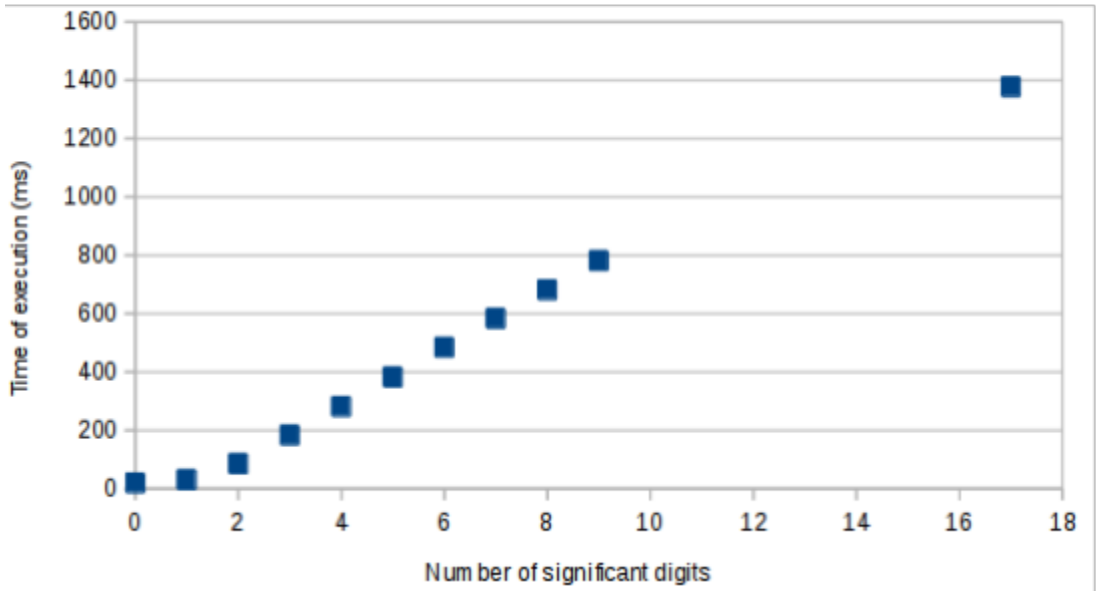
Correlation between memory usage and size of the grid is less straight forward. The correlation is positive up to grid size of 500, but it reduces later. Overall correlation coefficients for programs Tpdahp, Tpdohp, Tpfahp, and Twfahp are 0.75, 0.61, 0.64, and 0.45 respectively.

Chart 2: Grid size vs Memory usage

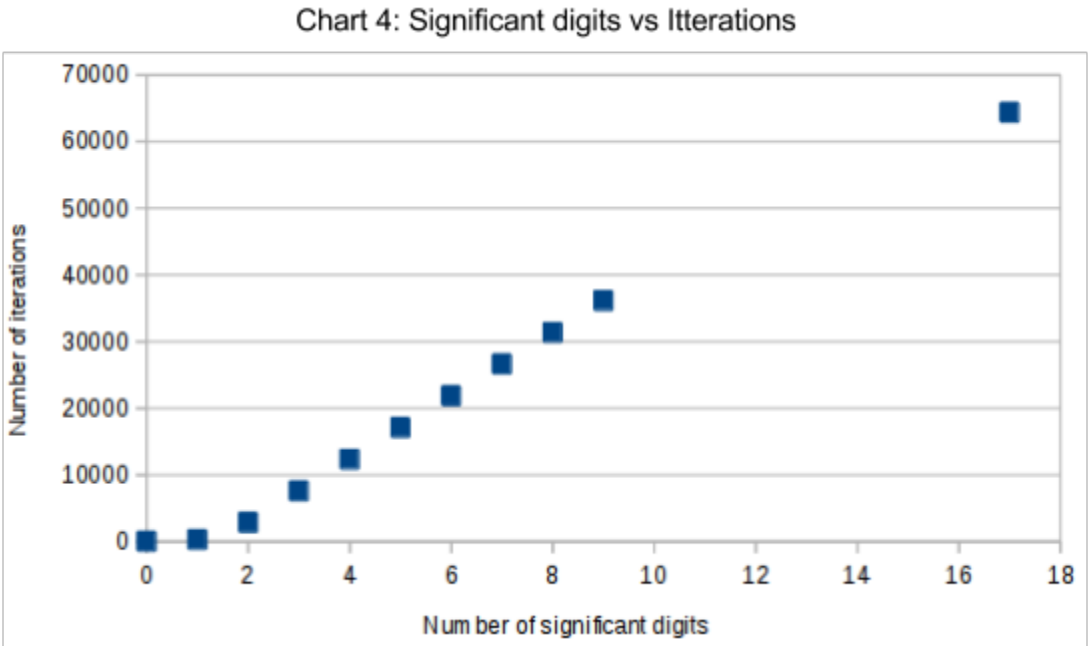


As shown on chart 3 below, the time of execution is directly and strongly correlated with precision, represented as a number of significant digits. The calculation of correlation coefficient proves this observation, resulting in the coefficient value equals to 0.99.

Chart 3: Significant digits vs Time of execution



Number of iterations as well strongly correlates with the number of significant digits, having the correlation coefficient equals to 0.99.



The very last trial where "float" type of variable was used to store the temperature values has produced the following results:

Program/Variable type	**Time of execution (ms)**	**Number of iterations**
Tpdahp	1377	64366
Tpfahp	703	22855

Discussion

The first chart shows that the time of execution increases proportionally to the size of the grid. This result can be expected, because the larger the grid the more time is required for a computer to calculate each value consequently.

Time of execution is lower for primitive data types and higher for objects. This is expected because primitive data types are stored on stack, rather than on heap, and are therefore easier to access. Regarding double and float it turned out that the variant of the program with double runs faster, even though it has better precision, which is 64 bits vs float's 32 bits. Double precision

(double) is actually faster than single precision (float) on some modern processors that have been optimized for high-speed mathematical calculations.

Twfahp variant of the simulation programs consumes the most amount of memory, comparing to other variants. This could be because of the boxing and unboxing as an extra step that has to be done to the values contained in the Double object type.

There is an anomaly in the data for memory usage. While the memory consumption grows with the size of the grid, this growth is not consistent. For example, memory usage for grid size of 200 is unusually higher than for 500 or 1000. This anomaly is not a mathematical one but instead is caused by the work of Java Garbage Collector, which gets triggered intermittently to clean objects, and release memory.

From the last two charts it can be seen that performance decreases with increasing of precision, and that precision does actually increases with increasing the relative change stopping criterion. Furthermore, the last trial of experiment shows that using “float” variable type produces much lower precision than when using “double”, because the number of iterations is significantly lower.

Conclusions

After conducting this study and analyzing its results we derived the following conclusions, which answers our research questions:

1) It was found that performance is directly affected by the grid size. The execution time is linearly proportional to the size of the grid.
2) It was found that performance is directly affected by the change in precision. The execution time is linearly proportional to the number of significant digits desired for temperature values.
3) UI hinges on code so it’s an expectation that the more UI elements that are required for the user experience the more lines of code the program will have.

4) It has been found that lattice design structure affects performance of the simulation. Four different design structures have been analyzed. It was found that using primitive data types and arrays in computationally intensive parts of the simulation is more effective in terms of performance than using objects or wrapped data types.

References

Pearson Correlation Coefficient Calculator

In-text: [1]

Bibliography: [1] Socscistatistics.com, 'Pearson Correlation Coefficient Calculator', 2014. [Online].

Available: http://www.socscistatistics.com/tests/pearson/. [Accessed: 21- Sep- 2014].