Parallel Algorithms
Classified Assignment

Wael Aljeshi*, Dan Demeter†, and Razvan Rosie‡

Department of Computing, Imperial College London

November 29, 2013

Mission Brief

A top secret computer system was discovered by extraction squad 429 in the nation of Nukehavistan. Highly detailed snapshots of the machine in operation were handed over, as well as a software implementation of the retrieved circuit board, residing in `mystery.o`. This report, addressed to the Director of Secret Services[1], details the methodologies of our mission and our findings.

1 Investigating the machine

The forensics team started by investigating the physical exterior of the machine, looking for clues which may allude to its function. This section discusses their methodologies and findings in detail.

1.1 Technical number enquires

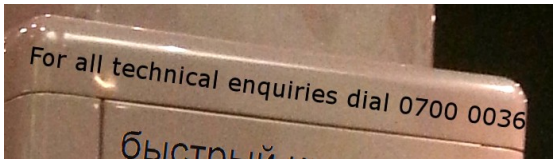

Figure 1: Technical enquiries phone number

The message in Figure 1 was prominently displayed near the top of the machine. A look up of historical phone directory records traced the number back to the *ORSA Journal on Computing*. A follow up visit was made to their customer services branch (Figure 2):

5521 Research Park Drive
Suite 200
Catonsville, Maryland 21228-4664
USA

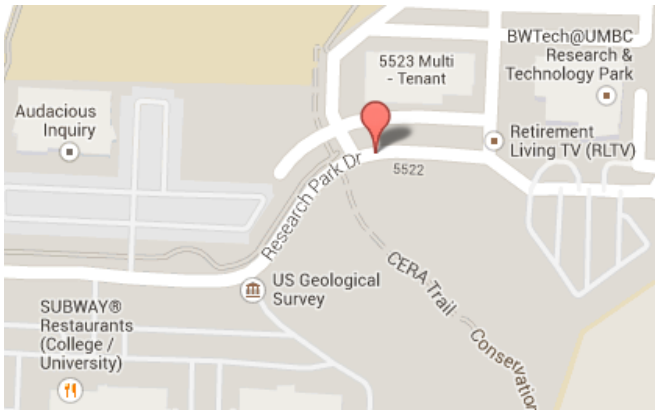

Figure 2: ORSA Journal on Computing customer services branch. Brunch was purchased from SUBWAY.

The team was then able to retrieve an excerpt[2] of a document *Numerical Inversion of Laplace Transforms of Probability Distributions*.

1.2 Cyrillic script

Some Cyrillic script was found on the machine, believed by our Business and Strategy experts to be used for marketing the machine in the Russian market.

An image of the script was analysed and translated electronically (Figure 3), yielding the rough English translation of *Fast inverter integrated functions*.

It is speculated that this marketing campaign was highly unsuccessful, as the marketing text consisted merely of a catchy phrase with no detailed description of operation [3]. Additionally, the provided help instructions were provided in a crude mixture of English and French, probably unhelpful for the Russian operators.

1.3 Geographic coordinates

The geographic coordinates $2.3328700E$ $48.8074800N$ were found inscribed on the machine. However, due

*wael.aljeshi10@imperial.ac.uk
†dan.demeter10@imperial.ac.uk
‡razvan.rosie10@imperial.ac.uk

[1] Sir Robert Hinchcliffe-Smythers-Smythe the 3rd

[2] An online version was independently found at `http://www.columbia.edu/~ww2040/Fall03/LaplaceInversionJoC95.pdf`

[3] Catchy marketing phrases are clearly an inferior marketing technique, and would never work — c.f. "Just do it" and "I'm lovin' it" for examples of such unsuccessful campaigns.

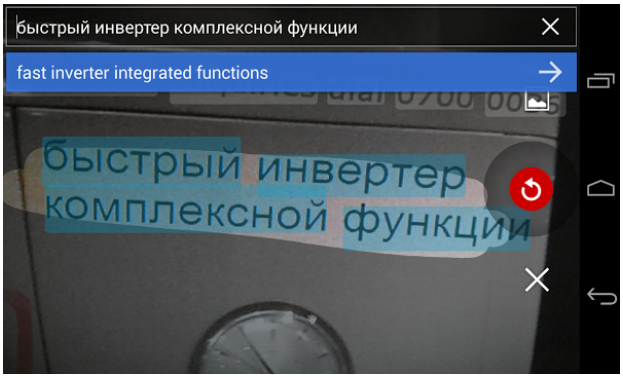

Figure 3: Cyrillic script found on the machine, along with the analysis results.

to a lack of 3G signal in the basement, the geographic team were unable to access the Internet and hence encountered difficulties in determining the whereabouts of the coordinates inscribed on the machine. The issue was promptly escalated to the CTO, who commissioned the purchase of two horses from a local supermarket. [4]

The location was found to be near the Laplace train station in France (Figure 4). Realising the mismatch in units due to the "inverse function" applied to the coordinates, the agents determined that the devious scheme was intended to be a very subtle hint that the machine may be performing an inverse Laplace transform. In order to cut costs, the horses were auctioned off at the nearby *Square du Serment de Koufra* for a profit of €7.89. This was sufficient to purchase baguettes for lunch.

Figure 4: Straight ahead: the main entrance of the Laplace train station. Right: the parked horses.

1.4 Portrait

A grayscale portrait was located on the central axis of the machine (Figure 5). As the portrait's CMYK band was much too narrow for our forensic experts' eyes, an image was instead captured and sent to the GlObal clOud-based Genealogy Locator E-service (Google). The subject was identified as Leonhard Euler,"a pioneering Swiss mathematician and physicist" (according to an omni-reliable, omni-malleable source [5]). This may hint that the machine employs Euler's methods for its computation.

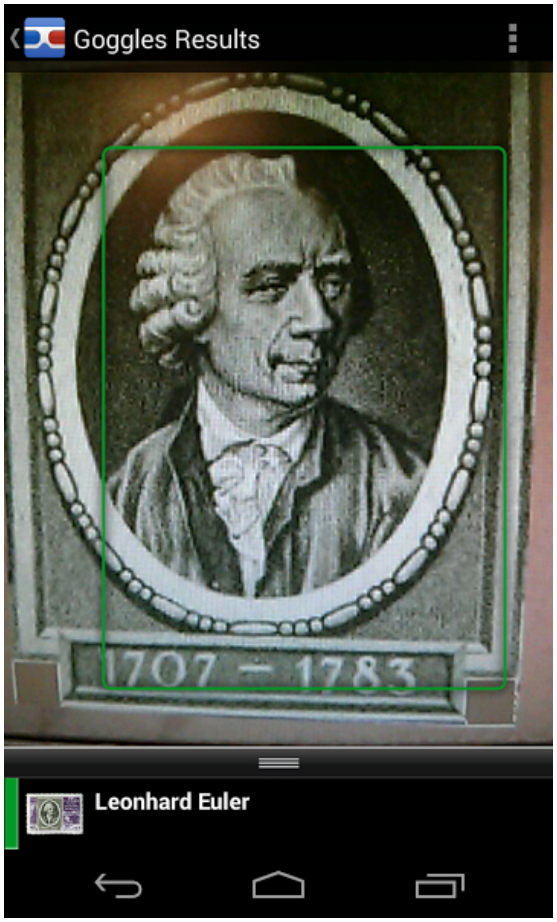

Figure 5: The result of the analysis on the portrait reveals the subject to be Leonhard Euler.

1.5 Wall clock and CRT Display

The machine panel has two seemingly, and deceivingly, unrelated components.

Firstly we have the wall clock (Figure 6), indicating the current time (as clocks do) to be 3:55, or more likely 15:55 given the sunlight in the background. Since Nukehavistan is located around $31N$ latitude (Figure 7), we are not dealing with a case of midnight sun[6].

Secondly we have the CRT display (Figure 8), which shows a graph of some function $f(t)$ over time. The graph has a positive peaks at $t = 3$ and a negative peak at $t = 11$.

After copious degrees of analysis, it was determined that the peaks in the graph are actually mirroring the

[4] Despite the falling price of petrol, it remains beyond our budget. As a plus, horses are also free to park.

[5] Wikipedia, the free encyclopedia

[6] `http://en.wikipedia.org/wiki/Midnight_sun`

[7] Image courtesy of The Onion `http://www.theonion.com/articles/us-intelligence-nukehavistan-may-have-nuclear-weap,1373/`

Figure 6: The wall clock located on the main panel of the machine

Figure 7: A map of the Republic of Nukehavistan [7]

hands of the clock, the positive peak representing the position of the hour hand (rounded down to the current hour), and the negative peak representing the position of the minute hand (rounded down to the nearest multiple of five). The fact that the t axis runs from 0 to 12 further supports this argument. Thus the graph in the image points at 3 (as is the current hour) and 11, translating to minutes $55 - 59$ (as is the current minute).

The circuit board is therefore used to plot a curve representing the current (12-hour) time (Figure 9).

2 Investigating the circuit

The next step was to investigate the pluggable circuit board, the process and results of which are discussed in this section.

2.1 Circuit board macro-analysis

The circuit board itself was identified as a product of AMD, housing a 500 MHz AMD Geode LX800 and 256 MB DDR DRAM (Figure 10). Such technology

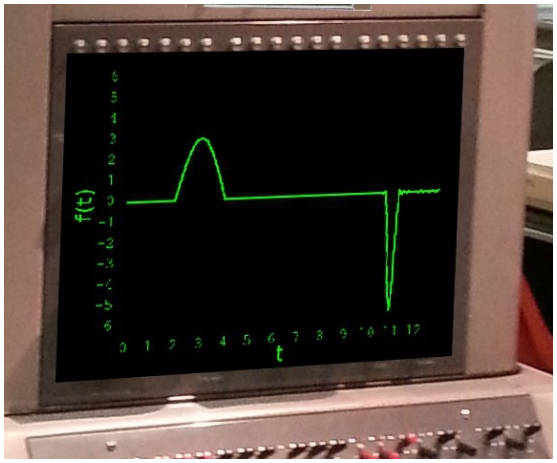

Figure 8: The CRT display showing a live graph in neon-ish green.

Figure 9: Fox news anchor Megyn Kelly commenting on the discovery.

was not believed to exist at the time, and it seems that great efforts have been devoted to keep this technology hidden.

This leaves investigating yet another set of geographic coordinates: $0.109167E$ $49.278611N$. Learning from their previous experience, the geographic team decided to book a flight this time. They, of course, did not learn from their *other* mistake, and hence spent quite some time travelling instead of taking the more sensible option of employing a parallel depth first search on the earth's map. The fact was reflected appropriately in their performance review. But we digress ...

After an expensive trip to France, it was determined that the site referred to "Ecole Beaumont en auge" at "Rue Pierre Simon Laplace Road" (Figure 11). The connection was made the following week, when the team realised that the circuit was meant to employ some form of Laplace transform.

2.2 Circuit board microanalysis

With the macro analysis out of the way, our experts studied the circuit board's functionality through means of the reconstructed `mystery.o` object file.

Figure 10: The identified circuit board

2.2.1 `strings` analysis

Executing `strings` analysis on `mystery.o` revealed the following hidden data:

```
(C) 2006 Svenska Aeroplan AB (SAAB)
Designad av Margarita González
Sampayo, Linkoping
```

The English translation:

```
(C) 2006 Swedish Aeroplane Company
Limited (SAAB) Designed by Margarita
González Sampayo, Linkoping
```

A background search revealed that Dr. Margarita Holmberg (González Sampayo was her maiden name) had centred her PhD thesis[8] around the Laplace Transform in electric circuits. Indeed, it seems that SAAB is mentioned twice in her thesis, in the context of a practical application of the Laplace transform:

> To solve problems working in Companies applying automatic control (for example, working in SAAB with aircraft dynamics) you use the Laplace transform.

[8] Engineering problem solving - The case of the Laplace transform as a difficulty in learning in electric circuits and as a tool to solve real world problems

Figure 11: Rue Pierre Simon Laplace Road - You will never find a more blessed hive of charm and harmony

2.2.2 `objdump` analysis

The symbol table of `mystery.o` was extracted using `objdump --syms`. This revealed a number of interesting library functions being used:

- Mathematical functions, often operating on complex numbers, such as `cexp`, `muldc3`, `log`, and others.
- Time-related functions, `time`, `strftime`, `strtol`, and `localtime`)

Whilst the mathematical functions were to be expected, the time-related functions came as a surprise to our agents. This discovery motivated further, more dynamic, investigation.

2.2.3 `ltrace` analysis

By running `ltrace` whilst running the team obtained the library calls used during the computation phase. The first and subsequent runs are almost identical, apart from an extra initialisation step. The initialisation step includes a call to `localtime`, protected by `cxa_guard` calls, which are used "to support thread-safe, one-time initialisation of function scope variables"[9] (Figure 12).

2.3 The Curious Case of `mystery.o`

Our agent from the *Hidden special unit for reverse analysis and decompilation* was able to successfully decompile the mysterious `mystery.o` file and write the equivalent C++ code. Further testing combined with the latest research in the field of *Reasoning about programs* confirmed that our code produces exactly the same results as the initial object file. Below are some of the pseudo-code function calls that are used to compute the final answer (see `code//mystery_impl.cpp` for full source):

[9] `http://www.opensource.apple.com/source/libcppabi/libcppabi-14/src/cxa_guard.cxx`

Initial call	Subsequent calls
time	time
__cxa_guard_acquire	
localtime	
__cxa_guard_release	
strftime	strftime
strftime	strftime
strtol	strtol
strtol	strtol
__muldc3	__muldc3
cexp	cexp
cexp	cexp
__divdc3	__divdc3
__muldc3	__muldc3
__muldc3	__muldc3
sincos	sincos
log	log
cexp	cexp
__divdc3	__divdc3
__muldc3	__muldc3
cexp	cexp
__muldc3	__muldc3

Figure 12: Trace of library calls for the initial and subsequent calls of the function `L()` . The extra library calls performed in the first function call are highlighted in red.

```
muldc0 = complex(arg1,arg2) *
    complex(arg1,arg2);
cexp0  = exp (complex(alpha * arg1, alpha *
    arg2);
cexp1  = exp (complex(beta * arg1, beta * arg2);
cexp2  = exp (complex(arg1 * -1 * (mins/5 +
    0.25), arg2 * -1 * (mins/5 + 0.25)));
[...]
multdc4 = complex(real(divdc1), imag(divdc1)) *
    complex(real(cexp3) , imag(cexp3));
result = compelx(real(muldc1) + real(muldc2) -
    real(muldc3) - real(muldc4), imag(muldc1) +
    imag(muldc2) - imag(muldc3) - imag(muldc4));
```

We would like to report some anomalies we have detected in the file, which we believe were put in place by the circuit designer in order to make the disassembly process more difficult:

2.3.1 Extra strings

We have noticed two ambiguous strings written at the beginning of the file:

```
(C) 2006 Svenska Aeroplan AB (SAAB)

Designad av Margarita González
Sampayo, Linkoping
```

They provide some insights into who wrote the initial software and the year it might have been written in.

2.3.2 Computing $y = x^{782768}$, where $x = 0.999\ldots$

Due to the fact that $0.999\ldots < 1$ and floating point numbers can't be represented exactly, the result of this exponentiation will be approximately 0.

2.3.3 Standard library functions: `sincos(x)` and `log(x + 1)`

The argument for the `sincos` function is y, the argument for the `log` function is $(y + 1)$, where y is the result of the multiplication discussed above. Because $y \approx 0$, `sincos(y)` returns $(0, 1)$ and `log(y+1)` returns 0.

2.3.4 Bogus while loops

During the research phase, our agent has identified four repetitive structures. One repetitive structure is used to compute $y = x^{782768}$ in a very inefficient way. The other two repetitive structures have the same implementation and they serve as iterators through the two initial strings defined in the program. Their main purpose is to compute the offset between a specific `ASCII` character `( 0x80 )` and the initial strings. Due to the fact that the strings are hard-coded, the results computed by both the strings will always be `35 ( 0x23 )` and `49 ( 0x31 )`. Lastly, the 4th structure is the `main` while loop. Its loop condition is:

$$49+35+sin(y)+cos(y)+log(1+y)-29-48+y > 20$$

Which simplifies to:

$$sin(y) + cos(y) + log(1 + y) > 13$$

Because y is a constant (0), $log(1+y)+sin(y)+cos(y)$ will be 1, so the loop condition $(8 > 20)$ will never be true.

Our agent is not sure whether these assembly artifacts and a lot of hard-coded values have been inserted in order to trick us or the compiler just generated general purpose code.

2.3.5 Compiler (ab)uses the stack base pointer `(RBP)`

During the analysis, it has been found that the `L()` function does not properly set the stack base pointer and instead it uses it as a general purpose register. This implies the fact that new variables are not created on the stack. The compiler is aware at compile time of the total number of variables used in the program and this is a known technique used by compilers in such situations.

2.3.6 Closing remarks

During the decompilation process our agent has used a test-driven approach, by writing scripts that would automate testing our computed function against the

original `mystery.o` one. We used the program **faketime** in order to generate 1440 tests cases: 1 test for each minute of the day. Then we used a parallel diff program that would call both our function and the original `L()` one with different parameters and report any different results. Part of the tools used by the agent were the GNU Debugger (gdb), with the peda plugin and the IDA program. No automated decompilation tools have been used (because there are none for 64-bit programs).

3 Serial Implementation

After agreeing that the machine is in fact an inverter of *Laplace* transforms, some time was spent investigating this mathematical concept. Directly, the transform can be given in the following way:

$$F(s) = \int_0^\infty f(t)e^{-st}\,\mathrm{d}t$$

Since we have established that we need to find the inverse transform, we notice the fact that:

$$f(t) = \int_0^\infty F(s)e^{st}\,\mathrm{d}t$$

To recap the data which we have gathered so far, the agents are facing the following problems:

- The analytical expression of the function F given by `mystery.o` is unknown.
- The `mystery.o` file only allows for evaluation of the function at some points passed as inputs.

Therefore, we have realised that we need to perform numerical inverse of $Inverse\ Laplace\ transform$. The paper offered us a valuable introduction to the problem that we were facing. The `EULER` algorithm was a starting point, and we proceed to the implementation step:

3.1 EULER Algorithm

The best description of the `EULER` algorithm can be summarised by the formula:

$$f(t) = \frac{10^{M/3}}{t}\sum_{k=0}^{2M}\eta_k Re(F(\frac{\beta_k}{t}))$$

Where:

$$\beta_k = \frac{Mln(10)}{3} + i\pi k$$

$$\eta_k = (-1)^k \cdot \xi_k$$

$$\xi_0 = \frac{1}{2}$$

$$\xi_k = 1, \text{ for } 1 \le k \le M$$

$$\xi_{2M-k} = \xi_{2M-k+1} + 2^{-M}\binom{M}{k}, \text{ for } 1 \le k < M$$

$$\xi_{2M} = \frac{1}{2^M}$$

The implementation of the algorithm (see `code/euler.cpp`) follows the procedure described above and can be summarised in the following manner:

- Select a size for the parameter M (for the current problem we use M as 30).
- Define an array `Times` to store the arguments where of the targeted function f.
- Construct an array of factors (ξ) that will act as factors in the summation of $F(\frac{\beta_k}{t})$.
  - Set $xi[0] = 1/2$.
  - Fill the elements on positions $1 \to M+1$ with 1's.
  - Fill the rest of the array with the factors of the binomial expansion of order $2M$.
- Precompute the constants described by the algorithm.
- For each point of time t, compute the sum described by the algorithm. This part of the code operates in an independent way (since we compute the sum for each point) and is suitable for a parallel implementation.
- Output the vector f_{st} containing the results of the evaluations.

Our serial implementation described the same graph as the one shown, with its global maxima pointing to the hour and the global minima pointing to the minute.

3.2 The Dubner-Abate Algorithm

After making sure that the `EULER` implementation works, the team decided to test other algorithms in order to compare the results and efficiency. We present here the *Dubner&Abate* algorithm, which is also computes the inverse of a *Laplace* transform as a *Fourier* series. The mathematics behind this algorithm is simpler because it does not require the computation of the binomial coefficients (which involve factorials) and may produce overflows.

In short, the *Dubner-Abate* algorithm will compute the following:

$$f(t) = \frac{e^{At}}{q}\sum_{k=1}^{M} Re(L(A + t\frac{ki\pi}{q}) \cdot cos(t\frac{k\pi}{q})) - \sum_{k=1}^{M} Im(L(A + t\frac{ki\pi}{q}) \cdot sin(t\frac{k\pi}{q}))$$

The serial implementation of the *Dubner-Abate* algorithm follows the logic described above.

- Select a size for the parameter M (for the current problem we use M as 200, for a reasonable approximation of the function f).
- Define an array `Times` to store the arguments where f will be approximated. We also precompute the constants described by the algorithm.
- For each point of time t, compute the *Dubner-Abate* algorithm's sum above. This part of the code operates in an independent fashion and (since we compute the sum for each point) and is suitable for a parallel implementation.
- Output the vector `f_ts` containing the results of the evaluations.

The fact that it independently computes $f(t)$ for every point makes this algorithm suitable for parallelisation.

4 Parallel Implementation

After we made sure that the serial implementation of the algorithm produces results that are accurate enough, we proceeded to the next step. A parallel version of the `EULER` algorithm was implemented using the *Message Passing Interface* (`MPI`).

In order to use the version of `MPI` installed on the labs machine, we have firstly created a host files, specifying the machines on which the program will run.

The in order, the `EULER` algorithm was parallelised in the following way (see `code/par-euler.cpp`):

1. Initialise the `MPI` execution environment by calling the `MPI_Init` function.
2. Call `MPI_Comm_rank` in order to determines the rank of the calling process in the communication system.
3. Call `MPI_Comm_size`, in order to determine how many processors are running the program.

Then, the root will generate the array of points `Times` where the function f will be evaluated. After the generation step is done, the root will broadcast the number of generated points (the actual size of `Times` vector) to the rest of the processors.

In order to do a `Scatter`, each processor should know the number of points it evaluates. These values are stored in the `sendcnts` array. If the size of `Times` will not evenly divide to the number of processors, the overflow will be distributed to the `sendcnts`.

At this stage, the displacement (from which to take the outgoing data to process) is also computed: this is simple since we know how many points are allocated per processor.

Finally, the `root` will create an array to receive the result of `MPI_Gatherv` in the array `f_ts`. The array will be printed to the standard output.

As a remark, for each expensive call, the method `CHECK` was used in order to verify the result of an `MPI` function call. If the result is not `MPI_SUCCESS`, the program will end. The function was applied on the calls to `MPI_Bcast`, `MPI_Scatterv` and `MPI_Gatherv`.

By using `MPI` we have made the execution time of computing `f(t)` decrease (the obvious advantage of using parallelism). However the inter-processor communication times have increased, but the total time was significantly improved, compared to the serial implementation. This enhancement can be best observed when the size of the vector of argument points for f is large and the algorithm executes on many processors.

5 Analysis Parallelise [10]

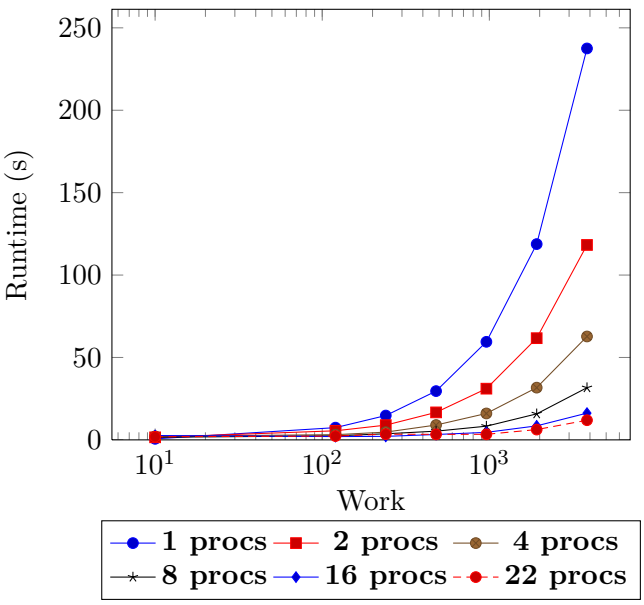

Figure 13: Comparison of runtime for different number of processors

We now compare the performance of our serial and parallel implementations of the machine. For this we run experiments with various the amount of work, i.e. the number of time points to be computed, from 10 points up to 3840 points. Additionally, each experiment is run serially, as well in parallel on a varying number of processors. The results of the experiments are shown in figure 16.

Figure 13 shows how the runtime of the computation varies with these two parameters. For a fixed number

[10] read: paralysis (or "paralylis")

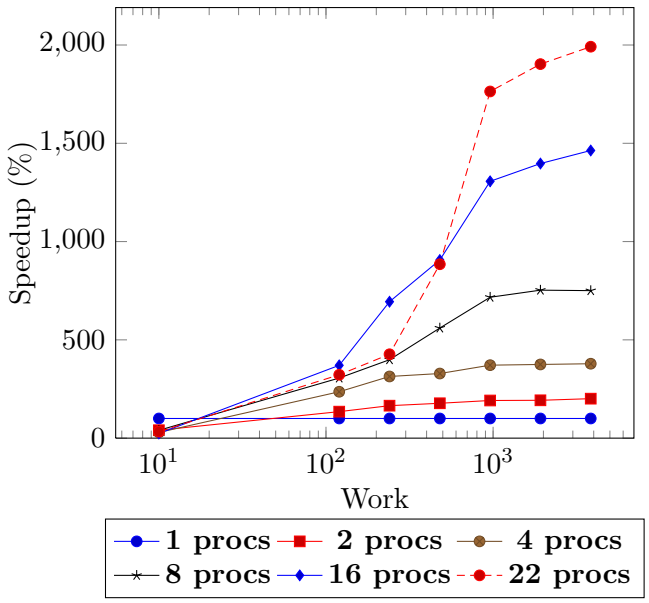

Figure 14: Comparison of speedup and efficiency for different number of processors

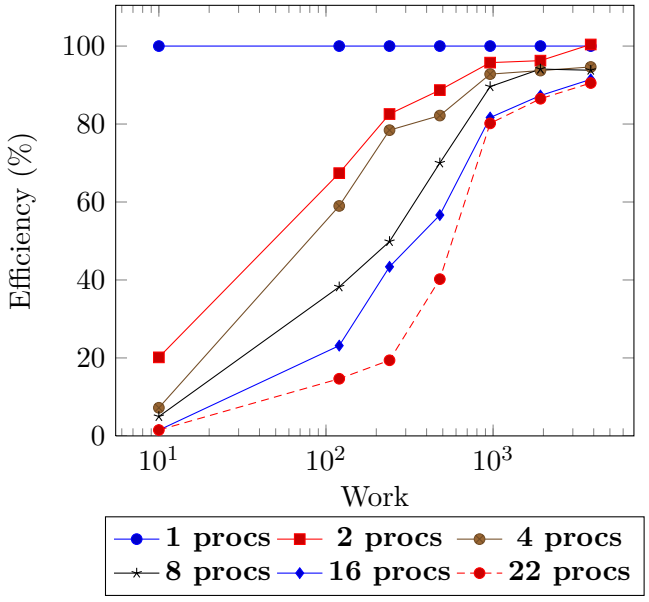

Figure 15: Comparison of speedup and efficiency for different number of processors

of processors, the run time of the algorithm increases as more work is added, as would be expected.

Figure 14 shows the speedup achieved through parallelisation. We note how the speedup attained improves greatly as we add more work. This corresponds to the declining contribution of communication overhead, more specifically that the communication overhead grows sub-linearly at most with respect to the program's fundamental running time.

Finally, figure 15 shows the efficiency of computation, that is, the percentage of useful work done per processor; it quantifies the contribution of parallelism overheads. We find that the efficiency increases with increased work, asymptotically approaching 100%, that is the serial algorithm.

	Number of processors					
Work	1	2	4	8	16	22
10	0.63	1.56	2.17	1.58	2.64	1.89
120	7.42	5.51	3.15	2.43	2	2.3
240	14.76	8.94	4.71	3.7	2.13	3.46
480	29.59	16.68	9	5.28	3.27	3.35
960	59.49	31.07	16.03	8.3	4.55	3.37
1,920	118.79	61.7	31.69	15.78	8.5	6.24
3,840	237.5	118.27	62.74	31.65	16.23	11.93

Figure 16: Measured run-times (in seconds) for different number of processors

6 Conclusion

After intense efforts, the agents have established the general purpose of the machine (a graph clock), as well as the functionality behind the mysterious circuit board.

During the technical phase of solving the tricky mission, agents have noticed the importance of the parallelism when applied to the problem of inverting a *Laplace* transform. Our solution presents the serial implementation of the $EULER$ algorithm as well as $Abate - Dubner$ algorithm. We have also parallelized the $EULER$ algorithm and presented the speedup achieved through parallelization and have discussed the resulting efficiency and how it relates to the problem-size.

Due to the mass benefits brought forth by parallelism we anticipate the existence of a cluster of such machines, perhaps in the cloud (figure 17). We therefore advise the secret service to further investigate this and locate the remaining instances (for example through random polling).

Figure 17: An artist's depiction of Nukehavistan's cloud [11]

[11] Image courtesy of computingcloudstorage.com `http://www.computingcloudstorage.com/wp-content/uploads/2013/04/Cloud-Server-Hosting.jpg`