Acceleration of Tandem Mass Spectrometry Analysis Software CoCoozo using Multi-core CPUs and Graphics Processing Units

Yasufumi Obata[1], Takashi Ishida[2], Tohru Natsume[3], and Yutaka Akiyama[2]
[1]Department of Computer Science, Faculty of Engineering, Tokyo Institute of Technology,
W8-76, 2-12-1 Ookayama, Meguro-ku, Tokyo 152-8550, JAPAN
[2]Department of Computer Science, Graduate School of Information Science and Engineering,
Tokyo Institute of Technology, W8-76, 2-12-1 Ookayama, Meguro-ku, Tokyo 152-8550, JAPAN
[3]Molecular Profiling Research Center for Drug Discovery,
National Institute of Advanced Industrial Science and Technology,
2-4-7 Aomi, Koto-ku, Tokyo 135-0064, JAPAN

Abstract— *Tandem mass spectrometry, a method involving multiple steps of mass spectral selection, is widely used in various biological fields. In recent years, steady improvements have been made with respect to speed, and the number of protein databases available for analysis has rapidly increased. Consequently, computational analysis has become the bottleneck in tandem mass spectrometry.*

To overcome this problem, we attempted to improve the tandem mass spectrometry analysis software CoCoozo. To accelerate the program, we improved the algorithm and also incorporated utilization of multi-core CPU and GPGPU. As a result of algorithm improvements, when all mass spectral data files had precursor data, we achieved 8.9-fold speedups compared with the original software. In addition, in the case of no precursor data, by using a 12-core CPU and a GPU card we achieved 18.1-fold speedups compared with the original software.

Keywords: Mass Spectrometry, MS/MS, CoCoozo, Multi-threading, GPGPU.

1. Introduction

Mass spectrometry is currently commonly used in proteomics research, a field of study in which the entire set of proteins expressed by a genome, cell tissue, or organism is examined [1]. Although various mass spectrometry methods have been developed, tandem mass spectrometry (MS/MS, MS^2) is the primary technique now used in many biological investigations, including research on cancer biomarkers [2], Alzheimer's disease [3], and protein-protein interactions [4].

In mass spectrometry analysis, target sample proteins or peptides are divided into several fragments whose masses are measured by a mass analyzer, with an analyzer outputting their spectra. Mass spectrometry analysis software is then used to identify the sample peptides or proteins based on these spectra. Tandem mass spectrometry, in which sample masses are measured in two or more steps, is currently in wide use. The advantage of tandem mass spectrometry is enabling the analysis of mixed protein samples. In two-step tandem mass spectrometry, the peptide ions fragmented from sample proteins during the first step are called precursors, and those fragmented from precursors during the second step are called fragments. Two-step tandem mass spectrometry generates mass spectral data for both precursors and fragments, and identifies sample proteins using both spectra. The mass spectral data contain mass-to-charge ratios (m/z) and collateral intensities of fragments and, in most cases, the precursor that is the source of the fragments.

Various software programs have been developed to analyze mass spectra from tandem mass spectrometry. To identify proteins in a sample, the software calculates the similarity between the spectral data and that of a protein in a database. MASCOT [5] is well known and widely used, but other software programs, such as SEQUEST [6], SpectraST [7], and CoCoozo have also been developed. Each program employs a different algorithm for database searching and similarity measurement, with each algorithm having different advantages and disadvantages with respect to speed and sensitivity. MASCOT, for example, uses a statistical evaluation algorithm [5], whereas SEQUEST uses a cross-correlation scoring algorithm [6]. For similarity measurement, SpectraST utilizes an inner product algorithm between measured and database mass spectral vectors [7].

CoCoozo is a mass spectrometry analysis software package developed at the National Institute of Advanced Industrial Science and Technology (AIST) of Japan and the Tokyo Institute of Technology. It features a unique error correction function for analyzing mass spectra with high precision. CoCoozo has performed consistently over the past several years of use in AIST projects.

In recent years, the speed and sensitivity of tandem mass spectrometry analyzers have steadily improved, with their throughput continually increasing. In addition, the number

Correspondence to: Yutaka Akiyama

of reference protein databases has steadily increased. Consequently, computational analysis is more time intensive than ever, and has become a bottleneck in mass spectrometry.

During the same time period, computer system performance has also been materially improved. Current computer systems have been enhanced by various acceleration technologies, including “multi-core CPU” and “General-Purpose computing on Graphics Processing Units” (GPGPUs). A graphics processing unit (GPU) was originally developed for processing graphics in the 1980s. GPU computational performance has increased dramatically over time, eventually overtaking that of CPUs. Consequently, GPUs have come to be used for general-purpose calculations rather than graphics processing, and this technique is now called “GPGPU”. A GPU has dozens or hundreds of streaming processors that are activated in parallel for calculations. GPGPU programing is difficult, requiring high parallel computing skills. To overcome this problem, several platforms have been developed to facilitate computational use of GPGPUs. At present, NVIDIA’s CUDA is the most substantial platform, and is widely used. Through the CUDA platform, programmers can make use of GPUs without having knowledge of GPU low-level instructions. As a result, GPGPU techniques have already been used in various applications, such as for astronomical calculations [8] and Fast Fourier Transform (FFT) [9].

In this paper, we report attempted enhancement of the computational speed of the tandem mass spectrometry analysis software package CoCoozo. To achieve this goal, we improved the algorithm and also incorporated multi-threading and the above-mentioned GPGPU-based acceleration technology at parts of similarity evaluation. As a result, when all mass spectral data files had precursor data, we achieved 8.9-fold speedups compared with the original software. In addition, in the case of no precursor data, by using a 12-core CPU we achieved 15.9-fold speedups with the original software. Moreover, by using a 12-core CPU and a GPU card we achieved 18.1-fold speedups compared with the original software.

2. CoCoozo

CoCoozo, which is mass spectrometry analysis software, has been already developed and used in several research fields. In this section, we briefly describe the algorithm of CoCoozo and discuss about its bottlenecks.

A flowchart of CoCoozo main processes is shown in Figure 1. For each precursor in a database, CoCoozo first checks whether or not a query data file includes precursor spectral data. If the query data file contains precursor spectral data, CoCoozo then performs a “precursor matching” process, which checks whether query precursor spectral data correspond to database precursor spectral data. If so, a “fragment matching” process is subsequently performed. If a query data file does not include precursor spectral data, the process goes directly to “fragment matching”, which calculates a similarity score between query fragment spectral data and fragment spectral data in a database. After fragment matching, a database precursor data entry has an assigned score based on the results of fragment matching. Finally, all precursors are ranked by their scores, with the highest-scoring precursors outputted as the analysis results.

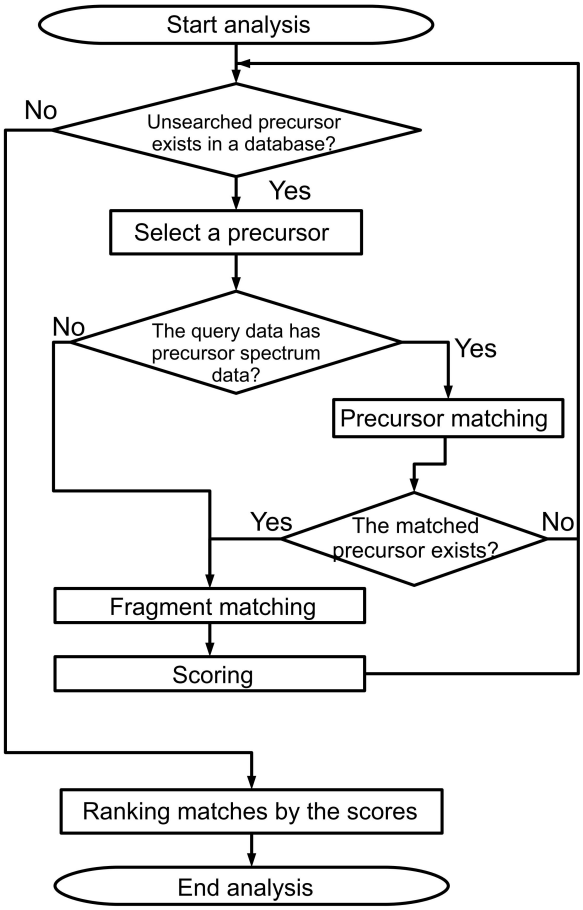

Fig. 1: CoCoozo Main Process Flowchart (for a mass spectral data)

During precursor and fragment matching processes, the matching algorithm judges whether m/z of a precursor or a fragment in a database matches, within a given tolerance, that of a query. Because the measured data sometimes includes error, the tolerance value is set based on the m/z value and a tolerance ratio parameter. One of the differences between precursor matching and fragment matching is the tolerance ratio parameter. In precursor matching, the ratio is fixed. In contrast, in fragment matching, the ratio varies depending on query spectrum intensity. When there are multiple spectra within a tolerance during fragment matching, the spectrum with the strongest intensity is judged as a “match”. The precursor matching process is therefore faster

than fragment matching.

If a data file has no precursor spectrum, the matching process takes much longer. This is because a large number of fragment matchings are needed in proportion to the number of precursor spectra in a database if a data file has no precursor spectrum. If, however, a data file contains a precursor spectrum, only a few fragment matchings are required because most of the precursors have already been filtered out based on the results of the precursor matching process.

2.1 Bottlenecks in CoCoozo

We profiled CoCoozo to locate the bottlenecks for two cases. The first case was one in which all mass spectral files included precursor spectral data (case of complete precursor data); the other case was one in which about 10% of mass spectral files were lacking precursor spectral data (case of incomplete precursor data).

The comparison between queries and database was found to be the dominant process in the entire execution time profile. This "matching" process targeting both precursors and fragments is the main process of the CoCoozo search algorithm. In addition, execution time was about 13 times longer in the case of incomplete precursor data compared with the case of complete precursor data. In the case of complete precursor data, the precursor matching comparison was the most dominant process in the entire execution time profile. On the other hand, fragment matching comparison was the most dominant process in the case of incomplete precursor data.

3. Methods

In this section, we introduce our newly proposed to accelerate the CoCoozo software using Multi-core CPUs and Graphics Processing Units.

3.1 Improvement of Matching Algorithm and Initialization Process

A similar matching algorithm is used for both precursors and fragments. To improve the process, the data is sorted that is replaced every comparison by m/z. In other words, we sort the data that is compared with the tolerance. Proteins are often diversely denatured, however, by post-translational modification (PTM), leading to a change in their masses. The way in which CoCoozo handles this PTM complicates sorting by m/z. We thus modified the program structure to improve the sorting. The new sorting algorithm allows matching to terminate when the sorted data exceed the tolerance upper limit. Because fragment matching has variable tolerance, the sort termination is based on a predefined maximum tolerance. In addition to early termination, the number of comparisons is decreased during precursor matching by skipping the data below the lower tolerance limit. This skip is especially effective in precursor matching because the number of comparisons with a given tolerance range is larger than in fragment matching. On the other hand, the skip is less effective in fragment matching because the number of comparisons with a given tolerance range is small.

In addition, we improved initialization of the variable for storing the score for each query. In the original initialization process, all of the scores are initialized regardless of whether or not they have changed. This guarantees that all scores are initialized when an analysis begins. Because initialization of all scores is redundant, scores that are unchanged since the last initialization are omitted.

3.2 Multithreading

When mass spectral files without precursor spectral data are included, the execution time materially increases. Based on profiling, fragment matching occupies about 85% of the entire execution time for the case of incomplete precursor data. Consequently, we apply a multithreading technique to fragment matching and scoring after fragment matching in cases in which an analysis target mass spectral data file lacks precursor spectral data. The two processes are consecutive. The one-time process targets database fragments created from the same precursor, and the consecutive process independence from other continuative processes, so the application of multithreading is relatively easy. In the multithreading part, each thread is in charge of matching fragments created from the same precursor and their scoring.

For multithreading implementation, we used the POSIX threads (pthread) library, a part of POSIX.1 [10] standardized by IEEE.

3.3 Acceleration by GPGPU

Even after improving the matching algorithm, fragment matching in the case of incomplete precursor data still occupied about 70% of the entire execution time. We therefore tried to introduce GPGPU to the fragment matching calculation. Each comparison between a fragment spectrum and one of the database entries in the fragment matching process is independent of the other comparisons, and each comparison is computationally not very intensive. The processes that follow the comparison are not independent, however, and the results of these processes depend on the results of other comparisons between database fragments created from the same precursor and query fragments. These should therefore be processed in serial, but serial processes are difficult to effectively execute on a GPU.

Consequently, we only applied the GPGPU technique to m/z and intensity comparisons. We parallelized each comparison on GPUs. Variable tolerance is inefficient on GPUs, however, because implementation of variable tolerance requires conditional statements, which results in CPU utilization scarcely decreasing. Because of this, a fixed tolerance based on maximum width ratio is used on the GPUs. In other words, preliminary selection is performed

on the GPU, with remaining matching executed on the CPU using results from the GPU.

In addition, we applied multithreading to the CPU processing that follows the GPU processing. The CPU processing corresponds to the original fragment matching and scoring. The one-time process targets database fragments created from the same precursor and the consecutive process independence from other continuative processes, so the application of multithreading is relatively easy.

For the GPGPU implementation, we used CUDA (Compute Unified Device Architecture), a platform for GPGPU provided by NVIDIA. Our software requires CUDA version higher than 2.3, and we used CUDA version 4.1 for the following experiments.

4. Results

4.1 Datasets and Database

We used 1,486 mass spectral data files as input queries. The files were in PKL format and had already been filtered. Because all of the files contained precursor spectrum data, we also prepared another dataset for checking the performance in the case of incomplete precursor data. In the second dataset, precursor spectral data was deleted from 149 randomly selected files, with the remaining 1,337 files identical to those in the original dataset. We used a database containing 38,415 proteins, 857,298 precursors, and 26,489,468 fragments, with lysyl endopeptidase (Lys-C) used for dividing a protein into precursors.

We allowed CoCoozo to search monovalent and divalent fragment ions, and to consider N-terminal acetylation.

4.2 Computing Environment

For this research, we used the TSUBAME2.0 supercomputer system at Tokyo Institute of Technology. Programs were executed on a thin node of TSUBAME2.0 with 12 CPU cores. Node specifications are shown in Table 1.

Table 1: Computing Environment

CPU	Intel Xeon 2.93 [GHz] (6 cores) x 2
Memory	54 [GB]
OS	SUSE Linux Enterprise Server 11 SP1
GPU	NVIDIA Tesla M2050
Compiler	gcc 4.3.4
MPI	OpenMPI 1.4.2
CUDA	CUDA 4.1 (64bit)
Profiler	Intel VTune Amplifier XE 2011

We used the UNIX "`time`" command to measure execution times and Intel VTune Amplifier XE 2011 for more detailed profiling.

4.3 Improvement of Matching Algorithm and Initialization Process

Figure 2 shows execution time results when all of the mass spectral data files include precursor spectral data (case of complete precursor data). CoCoozo with the improved algorithm is approximately 8.9-fold faster than the original version. In particular, a precursor-matching step is about 65.3-fold faster than that of the original, and a fragment-matching step is approximately 2.5-fold faster. With respect to score initialization, the improved version is approximately 483.1-fold faster than the original, equivalent to other short processes.

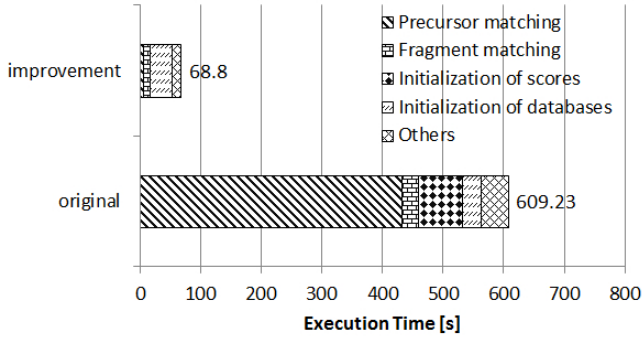

Fig. 2: Result of "Improvement of Matching Algorithm and Initialization Process" (execution time) in the case of complete precursor data.

Table 2: Results of Improvements in the case of complete precursor data

	times [sec]	speedup
Original	609.23	
- Precursor-matching	443.0	
- Fragment-matching	27.61	
- Score Initialization	72.46	
Improvement of Algorithm	68.80	8.9-fold
- Precursor-matching	6.63	65.3-fold
- Fragment-matching	11.08	2.5-fold
- Score Initialization	0.15	483.1-fold

These results demonstrate the magnitude of the improvements arising from the revised algorithm in the case of complete precursor data. Table 2 summarizes the results of improvements in the case of complete precursor data.

4.4 Multithreading

Figure 3 shows execution time as a function of number of threads when about 10% of mass spectral data files lack precursor spectra (case of incomplete precursor data). As seen in the figure, even in the one-thread case, execution is much faster than in the original version, because the multi-threaded version used an improved matching algorithm. CoCoozo with 12 threads is approximately 5.3-fold faster than CoCoozo with 1 thread; the increased speed is less than 12-fold because multi-threaded processing is only applicable to certain parts of the entire program. Another reason is concurrency of threads. As measured by Intel Vtune Amplifier, the peak concurrency is 6 threads even on

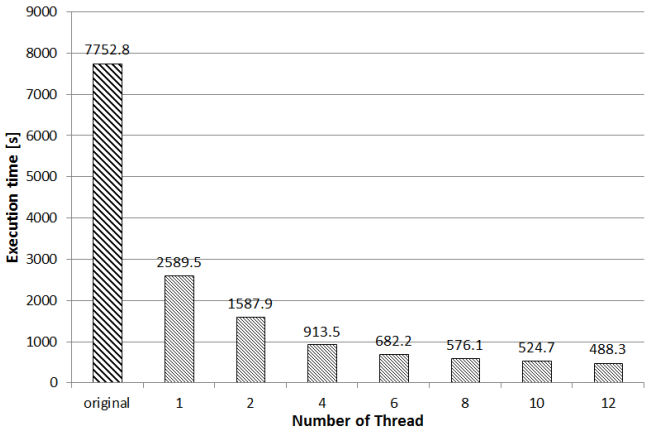

Fig. 3: Result of "Multithreading" (execution time) in the case of incomplete precursor data.

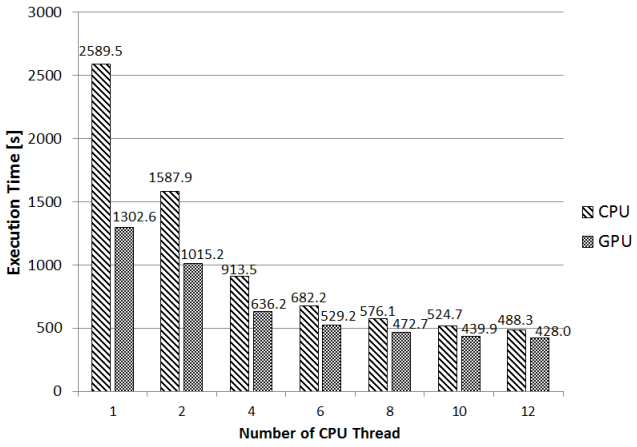

Fig. 4: Result of "Acceleration by GPGPU" (execution time) in the case of incomplete precursor data.

a 12 CPU core system: 12 threads cannot be simultaneously used, and several threads are often idle.

Finally, compared with the original version, CoCoozo with the improved matching algorithm and running with 12 threads is approximately 15.9-fold faster.

4.5 Acceleration by GPGPU

Figure 4 shows CoCoozo execution time with GPGPU for the case of incomplete precursor data. In the figure, the data represented by bars labeled "CPU" is the same as in Figure 3. CoCoozo with GPGPU is approximately 2.0-fold faster than CoCoozo with the improved algorithm but without GPGPU, and approximately 6.0-fold faster than the original version. In particular, a fragment-matching step with GPGPU is approximately 13.8-fold faster than that with the improved matching algorithm. CoCoozo with GPGPU and 12 threads is approximately 3.0-fold faster than CoCoozo with GPGPU and 1 thread. The reason for the low efficiency

Table 3: Results of Improvements in the case of incomplete precursor data

	times [sec]	speedup
Original	7752.82	
Improvement of Algorithm	2589.52	3.0-fold
Multithreading (12-thread)	488.30	15.9-fold
GPGPU	1302.57	6.0-fold
Multithreading (12-thread) & GPGPU	427.97	18.1-fold

gain is the same as above. When concurrency is measured with Vtune Amplifier, peak concurrency is 3 threads, and with parallel execution is scarcely over 9 threads. The deterioration of concurrency is caused by the use of GPGPU, as it assists in fragment matching comparisons. Because there is less opportunity for multithreading processes when using GPGPU, more threads are idle. Finally, CoCoozo with GPGPU and 12 threads is about 18.1-fold faster than the original version.

Table 3 summarizes the results of improvements in the case of incomplete precursor data.

5. Discussion

Although analysis results are almost unchanged following improvements, a subtle difference was noted: the values of some scores are different. This difference does not affect the substance of the results, and thus the results after implementation of improvements are not distinguishable from the original ones. The difference is caused by results from fragment matching changing slightly because of data sorting, but this only appears when some peaks having the exactly same intensities are sorted and the matching order of the peaks change from the original order. We believe this change seldom occurs and has little or no effect.

In the case of complete precursor data, database initialization dominates program execution time, representing over 50% of the entire elapsed time. Because initialization is only executed once at the beginning of the program, however, it is not a very serious problem, even when very large query data files are inputted.

6. Conclusions

We have enhanced the tandem mass spectrometry analysis software CoCoozo in three ways: through improved matching and initialization algorithms, multithreading, and GPGPU. When mass spectral data files all contain precursor spectrum data, CoCoozo with the improved algorithm achieves an 8.9-fold speedup compared with the original version. In cases where 10% of mass spectral data files lack precursor spectrum data, CoCoozo with the improved algorithm is 3.0-fold faster than the original. In addition, the multithreading version of CoCoozo with 12 CPU cores achieves a 15.9-fold speedup, and the GPGPU version of

CoCoozo with 1 GPU and 12 CPU cores is 18.1-fold faster than the original.

In this research, we applied multithreading programming and GPGPU only to the case of incomplete precursor data. The case of incomplete precursor data does not often occur in practice, however. Consequently, the application of multithreading programing and GPGPU to the case of complete precursor data is an important focus of future work.

Acknowledgements

We are deeply grateful to Mr. Katsuyuki Koike, Mr. Hideo Kusano, Mr. Tomohisa Hatta, the members of the developing team of original version of CoCoozo, for their insightful comments and suggestions. We also thank Mr. Yasuhiro Fujihara, a main programming staff of the original version of CoCoozo, for his generous support.

References

[1] W. P. Blackstock, M. P. Weir, “Proteomics: quantitative and physical mapping of cellular proteins.”, *Trends Biotechnol.*, vol. 17, No. 3, pp. 121-127, 1999.

[2] E. P. Diamandis, “Mass spectrometry as a diagnostic and a cancer biomarker discovery tool: opportunities and potential limitations.”, *Mol Cell Proteomics.*, vol. 3, no. 4, pp. 367-378, 2004.

[3] D. C. German, P. Gurnani, A. Nandi, H. R. Garner, W. Fisher, R. Diaz-Arrastia, P. O’Suilleabhain, K. P. Rosenblatt: “Serum biomarkers for Alzheimer’s disease: proteomic discovery.”, *Biomed Pharmacother.*, vol. 61 no. 7, pp. 383-389, 2007.

[4] A. C. Gavin, K. Maeda, S. Kühner, “Recent advances in charting protein-protein interaction: mass spectrometry-based approaches.”, *Curr Opin Biotechnol.*, vol. 22 no. 1, pp. 42-49, 2011.

[5] D. N. Perkins, D. J. Pappin, D. M. Creasy, J. S. Cottrell, “Probability-based protein identification by searching sequence databases using mass spectrometry data”, *Electrophoresis.*, vol. 20, no. 18, pp. 3551-3567, 1999.

[6] J. K. Eng, A. L. McCormack, J. R. Yates, III, “An Approach to Correlate Tandem Mass Spectral Data of Peptides with Amono Acid Sequences in a Protein Database”, *J. Am. Soc. Mass Spectrom*, vol. 5, no. 11, pp. 976-989, 1994.

[7] H. Lam, E. W. Deutsch, J. S. Eddes, J. K. Eng, N. King, S. E. Stein, R. Aebersold, “Development and validation of a spectral library searching method for peptide identification from MS/MS”, *Proteomics*, vol. 7, no. 5, pp. 655-667, 2007.

[8] Hubert Nguyen, *GPU Gems 3*, Boston, U.S.A.: Addison-Wesley Professional, 2007.

[9] N. K. Govindaraju, B. Lloyed, Y. Dotsenko, B. Smith, J. Manferdelli, “High Performance Discrete Fourier Transforms on Graphics Processors”, *the 2008 ACM/IEEE conference on supercomputing*, pp. 1-12, 2008.

[10] *Information Technology - Portable Operating System Interface (POSIX) - Part1 : System Application Program Interface (API) [C Language]*, IEEE, Inc., 1996.