ACCELERATED LEARNING OF GAUSSIAN PROCESS MODELS

Bojan Musizza[1], **Dejan Petelin**[1], **Juš Kocijan**[1,2]

[1]Jožef Stefan Institute
Jamova 39, Ljubljana, Slovenia
[2]University of Nova Gorica
Vipavska 13, Nova Gorica, Slovenia

dejan.petelin@ijs.si(Dejan Petelin)

Abstract

The Gaussian process model is an example of a flexible, probabilistic, nonparametric model with uncertainty predictions. It offers a range of advantages for modelling from data and has been therefore used also for dynamic systems identification. One of the noticeable drawbacks of the system identification with Gaussian process models is computation time necessary for modelling. The modelling procedure involves the inverse of covariance matrix which is as large as the length of input samples vector. The computation time for this inverse regardless of the use of efficient algorithm is rising with the third power of input data number. Intensive research is going on for finding algorithms that would accelerate the training of Gaussian process models. The purpose of this paper is to show approach from the used hardware point of view. The assessment of computational efficiency of two different hardware platforms for GP model identification are given in the paper. These are: single core personal computer and personal computer with graphic card used for computations. The assessment has been done with comparison of computational load on a toy case study of nonlinear dynamic system identification. The assessment reveals that the parallel computation solutions are efficient for larger amount of data when the initial and communication overhead of parallel computation becomes sufficiently small part of the whole process.

Keywords: Gaussian process models, Dynamic system models, System identification, Simulation.

Presenting Author's Biography

Dejan Petelin received the M.Sc. degree in computer science and informatics from the Faculty of Computer Science and Informatics, University of Ljubljana. He is currently a Ph.D. student at the Department of Systems and Control, Jozef Stefan Institute in Ljubljana. His main research interests are machine learning methods and their application for dynamic systems modelling.

1 Introduction

Gaussian process (GP) models form a new, emerging complementary method for nonlinear system identification. GP model is a probabilistic nonparametric black-box model. It differs from most of the other frequently used black-box identification approaches as it does not try to approximate the modelled system by fitting the parameters of the selected basis functions but rather searches for the relationship among measured data. Gaussian processes models are closely related to approaches such as Support Vector Machines and specially Relevance Vector Machines [7]. Because GP model is a Bayesian model, the output of Gaussian process model is a normal distribution, expressed in terms of mean and variance. Mean value represents the most likely output and the variance can be viewed as the measure of its confidence. Obtained variance, which depends on amount of available identification data, is important information distinguishing the GP models from other non-bayesian methods. Gaussian process can be used for model identification when data are heavily corrupted with noise, when there are outliers or gaps in the input data. Another useful attribute of GP model is the possibility to include various kinds of prior knowledge into the model, e.g. local models, static characteristic, etc.

Applications of the GP model for the identification of dynamic systems are presented in e.g. [1], [2] and [4].

A noticeable drawback of the system identification with Gaussian process models is computation time necessary for modelling. Gaussian process regression involves several matrix computations which load increases with the third power of the number of input data, such as matrix inversion and the calculation of the log-determinant of used covariance matrix. This computational greed restrict the number of training data, to at most a few thousand cases.

To overcome the computational limitation issues and make use of the method also for large-scale dataset application, numerous authors have suggested various sparse approximations. Authors of [6] have provided a unified view of sparse Gaussian process approximation, which includes a comparison of work published by various authors. Common to all these approximation methods is that only a subset of the variables is treated exactly, with the remaining variables given some approximate, but computationally cheaper approach.

The purpose of this paper is to approach the computation problem from utilised hardware technology point of view. This approach might seem inefficient, but it is undoubtedly effective. The assessment of computational efficiency of four different hardware platforms for GP model identification that are affordable for most of research groups is the topic of this paper.

The paper is composed as follows. The next section will briefly describe the modelling of dynamic systems with Gaussian process models. The description of dynamic systems simulation will follow in Section 3. Hardware configurations to be assessed are given in Section 4 and the assessment with a benchmark case study is described in the Section 5. Conclusions are given at the end of paper.

2 Modelling of Dynamic Systems with Gaussian Processes

A Gaussian process is an example of the use of a flexible, probabilistic, non-parametric model with uncertainty predictions. Its use and properties for modelling are reviewed in [7].

A Gaussian process is a collection of random variables which have a joint multivariate Gaussian distribution. Assuming a relationship of the form $y = f(\mathbf{x})$ between an input $\mathbf{x}$ and output y, we have $y_1, \ldots, y_n \sim \mathcal{N}(0, \Sigma)$, where $\Sigma_{pq} = \mathrm{Cov}(y_p, y_q) = C(\mathbf{x}_p, \mathbf{x}_q)$ gives the covariance between output points corresponding to input points $\mathbf{x}_p$ and $\mathbf{x}_q$. Thus, the mean $\mu(\mathbf{x})$ (usually assumed to be zero) and the covariance function $C(\mathbf{x}_p, \mathbf{x}_q)$ fully specify the Gaussian process. Note that the covariance function $C(.,.)$ can be any function having the property of generating a positive definite covariance matrix.

Covariance function $C(\mathbf{x}_p, \mathbf{x}_q)$ can be interpreted as a measure of distance between input points $\mathbf{x}_p$ and $\mathbf{x}_q$. For systems modelling it is usually composed from two main parts:

$$C(\mathbf{x}_p, \mathbf{x}_q) = C_f(\mathbf{x}_p, \mathbf{x}_q) + C_n(\mathbf{x}_p, \mathbf{x}_q) \qquad (1)$$

where C_f represents functional part and describes the unknown system we are modelling and C_n represents noise part and describes the model of noise.

A common choice is

$$C_f(\mathbf{x}_p, \mathbf{x}_q) = v_1 \exp\left[-\frac{1}{2}\sum_{d=1}^{D} w_d (x_{dp} - x_{dq})^2\right] + \delta_{pq} v_0, \qquad (2)$$

where $\boldsymbol{\Theta} = [w_1 \ldots w_D \;\; v_0 \;\; v_1]^T$ are the 'hyperparameters' of the covariance functions, D is the input dimension and $\delta_{pq} = 1$ if $p = q$ and 0 otherwise. The square exponential covariance function represents smooth and continuous functional part and the constant covariance function represents the noise part when it is presumed to be the white noise. Other forms of covariance functions suitable for different applications can be found in [7]. For a given problem, the parameters are learned or identified using the data at hand. After the learning, one can use the w parameters as indicators of 'how important' the corresponding input regressors are: if w_d is zero or near zero it means that the inputs in dimension d contain little information and could possibly be removed.

Consider a set of N D-dimensional input vectors $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]$ and a vector of output data $\mathbf{y} = [y_1, y_2, \ldots, y_N]^T$. Based on the data $(\mathbf{X}, \mathbf{y})$, and given

a new input vector $\mathbf{x}^*$, we wish to find the predictive distribution of the corresponding output y^*. Unlike other models, there is no model parameter determination as such, within a fixed model structure. With this model, most of the effort consists in *tuning* the parameters of the covariance function. This is done by maximisation of the log-likelihood

$$\begin{aligned}\mathcal{L}(\boldsymbol{\Theta}) &= \log(p(\mathbf{y}|\mathbf{X})) \\ &= -\frac{1}{2}\log(\mid \mathbf{K} \mid) - \frac{1}{2}\mathbf{y}^T\mathbf{K}^{-1}\mathbf{y} - \frac{N}{2}\log(2\pi)\end{aligned} \tag{3}$$

where $\boldsymbol{\Theta}$ is the vector of hyperparameters and $\mathbf{K}$ is the $N \times N$ training covariance matrix. The number of parameters to be optimized is small ($D + 2$, see equation (2)), which means that optimization convergence might be faster and that the 'curse of dimensionality' so common to black-box identification methods is circumvented or at least decreased.

The described approach can be easily utilized for regression calculation. Based on training set $\mathbf{X}$ a covariance matrix $\mathbf{K}$ of size $N \times N$ is determined. As already mentioned, the aim is to find the distribution of the corresponding output y^* at some new input vector $\mathbf{x}^* = [x_1(N+1), x_2(N+1), \ldots, x_D(N+1)]^T$.

For a new test input $\mathbf{x}^*$, the predictive distribution of the corresponding output is $y^*|(\mathbf{X}, \mathbf{y}), \mathbf{x}^*$ and is Gaussian, with mean and variance

$$\mu(\mathbf{x}^*) = \mathbf{k}(\mathbf{x}^*)^T\, K^{-1}\, \mathbf{y}, \tag{4}$$

$$\sigma^2(\mathbf{x}^*) = k(\mathbf{x}^*) - \mathbf{k}(\mathbf{x}^*)^T\, \mathbf{K}^{-1}\, \mathbf{k}(\mathbf{x}^*), \tag{5}$$

where $\mathbf{k}(\mathbf{x}^*) = [C(\mathbf{x}^1, \mathbf{x}^*), \ldots, C(\mathbf{x}^N, \mathbf{x}^*)]^T$ is the $N \times 1$ vector of covariances between the test and training cases, and $k(\mathbf{x}^*) = C(\mathbf{x}^*, \mathbf{x}^*)$ is the covariance between the test input and itself.

Gaussian processes can, like other machine learning methods, e.g. neural networks, be used to model static nonlinearities and can therefore be used for modelling of dynamic systems [1], [2], [4] if delayed input and output signals are fed back and used as regressors. In such cases an autoregressive model is considered, such that the current output depends on previous outputs, as well as on previous control inputs.

$$\begin{aligned}\mathbf{x}(k) &= [y(k-1), y(k-2), \ldots, y(k-L), \\ &\quad u(k-1), u(k-2), \ldots, u(k-L)]^T, \\ y(k) &= f(\mathbf{x}(k)) + \epsilon,\end{aligned} \tag{6}$$

where k denotes the consecutive number of data sample. Let $\mathbf{x}$ denote the state vector composed of the previous outputs y and inputs u up to a given lag L, and ϵ is white noise.

When only the mean values of the model predicted values are fed back the simulation was named 'naive'. This is the sort of simulation that will be used for the case study simulation in the paper. However, to get a more realistic picture of the dynamic model multi-step-ahead prediction we could take account of the uncertainty of the future predictions, which provide the 'inputs' for estimating further means and uncertainties. More details on such kind of simulation can be found in [4].

As can be seen from the presented relations, the obtained model not only describes the dynamic characteristics of nonlinear system, but also provides information about the confidence in these predictions by means of prediction variance. The Gaussian process can highlight areas of the input space where prediction quality is poor, due to the lack of data, by indicating the higher variance around the predicted mean.

The algorithms for identification or training of Gaussian process dynamic model and its simulation is in our case pursued with a set of routines [3] that are based on Matlab programme package and are upgrade of the GPML toolbox [7] for machine learning with Gaussian processes.

3 Acceleration with various hardware configurations

In order to show how the computation problem of the GP modeling and prediction can successfully be tackled with the use of currently avaliable hardware, we describe the computing architectures that were used in our tests.

- **Single core computer (SC).** Here the standard PC equipped with Intel Pentium 4 processor was used. The processor is a one-core version. The processor runs @3.2 GHz and uses DDR2 memory. This configuration was primarily used to set the base level to which all other results could be compared.

- **Personal computer with GPU.** This configuration includes high performance PC with the Intel i7 860 Processor running @2.8GHz. The computer is equipped with the NVIDIA Tesla C1060 computing processor board, which is build around the NVIDIA Tesla T10 graphics processing unit (GPU). The processor includes 240 processor cores running @1.296 GHz, with memory interface running @800 MHz. The board contains 4GB of GDDR3 memory capable of 102 GB/s peak bandwidth. In this configuration the board is capable of peak performance of 933 GFLOP/s.

  The GPU processor evolved from the needs of the 3D graphics intensive applications. This need dictated the design of the processor so that more transistors were dedicated to the data processing rather than to the control and data caching as in regular CPU. Next, the processor was designed to be able to execute data-parallel algorithms, consequently, the GPU architecture is sometimes described as a SIMD architecture.

The current tools for GP modelling and prediction are packed as a custom Matlab toolbox that is optimized for single processor architecture. The code is mainly

written in Matlab with time critical section written in C mex files.

We have written a small benchmark program based on the GP toolbox, which tests execution times of a typical GP training and prediction cycle. The program gathers computation times in relation to the size and dimension of input data. This approach gives the view on the impact different architectures have on the computation time from user's perspective.

The fact that the GP toolbox is written for single core configuration, we had to implement the critical functions of the toolbox to be able to run on GPU. The functions had to be written in a way not to disturb the current structure of the toolbox.

The NVIDIA GPU comes with the API, which had to be integrated into Matlab's mex functions. Additionally several other programming libraries had to be used in order to implement time critical operations such as matrix factorization and inverse. For this purpose we used CUBLAS, which is the NVIDIA's implementation of the popular library BLAS. Next, we used CULA Premium libraries for the implementation of time critical Cholesky factorization. Additionally, several other operations had to be implemented, with custom code and by using special functions called kernels, which when called are executed N times in parallel in N different threads.

By using C code in mex files only minor changes had to be performed on the user side of the GP toolbox in order to use the complex high-performance GPU architecture.

The GPU is currently a low-cost High Performance Computing alternative. With its intrinsic parallel structure allows significant speedup in comparison to the single processor architecture. Although it is relatively easy to setup and perform basic operations it quickly becomes more complex when dealing with more demanding numerical problems. Additionally, special care must be taken, when performing memory operations. These can present a significant bottleneck when insufficient care is taken in designing the algorithm. Therefore all the data transfers are performed only at the begining and the end of the algorithm.

4 Case study

The following example is intended for assessment of the potential of various hardware configurations for accelerating learning and simulation of Gaussian process model of dynamic systems. Consider the nonlinear dynamic system [5] described by

$$y(k+1) = \frac{y(k)}{1+y^2(k)} + u^3(k) + \epsilon \qquad (7)$$

where u is system's input signal, y is system's output signal, ϵ is white noise of normal distribution and standard deviation 0.05 that contaminates the system response and the sampling time is one second. The nonlinearity in the region of interest for the benchmark system is depicted in Figure 1.

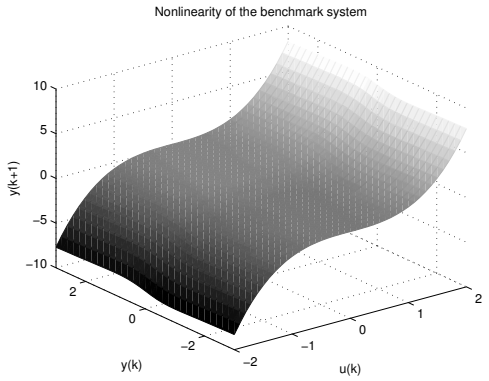

Fig. 1 The nonlinearity of the dynamic nonlinear system benchmark used for the case study

The segment of input signal used for system identification is given in Figure 2. The input signal is generated with random generator with uniform distribution of magnitudes in the interval [-1.5,1.5] and sampling time 10 s.

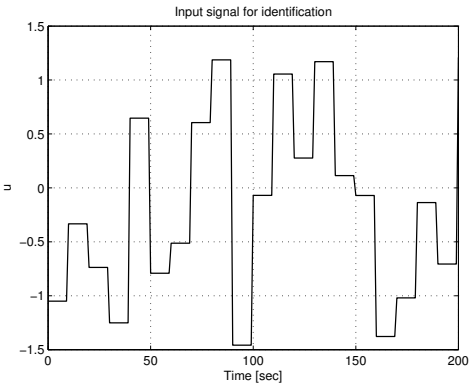

Fig. 2 Input signal used for identification - the segment of first 200 samples

The segment of response of the system corresponding to the input signal segment in Figure 2 is given in Figure 3.

The segment of input signal used for model validation is given in Figure 4. The input signal for validation is generated with random generator in the same way as identification, but with different sequence to obtain signal that is different from the one used for identification.

The segment of response of the system corresponding to the input signal segment in Figure 4 is given in Figure 5.

Modelling was pursued with both hardware configurations named in the previous section: single core personal computer and personal computer with graphic card used for computations.

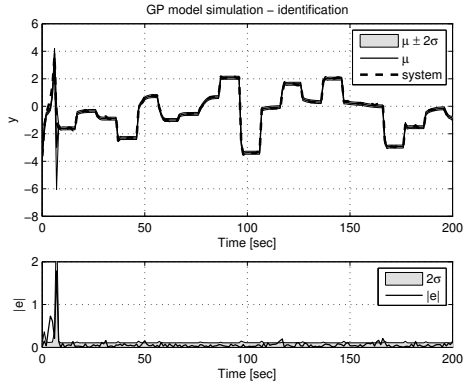

Fig. 3 Gaussian process model response on identification signal and comparison with the system response (top figure) and absolute value of model residuals with 95% confidence band (bottom figure) - the segment of first 200 samples

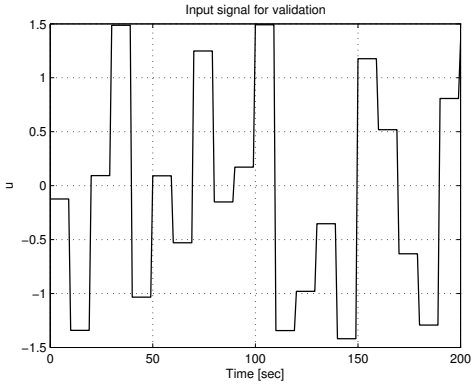

Fig. 4 Input signal used for validation - the segment of first 200 samples

The comparison of identification computation times with different input data dimensions are given in Figure 6.

For the identification the computation of a matrix inversion is needed. Its time complexity rises with third power of amount of data. It can be seen that the identification for relative small amount of data (in our case smaller than approximately 250 data) is faster on single core computer. For larger amount of data the identification is considerably faster on GPU - it's factor rises with power (note that scale of Figure 6 is logaritmic). In our case the identification on GPU for 1000 data is approximately 1.4 times faster and approximately 2.5 times faster for 2000 data than on SC computer.

Relatively poor performance for smaller input data sizes is due to initial memory initialisation required by a parallel computation and data transfer. These operations are needed only at the begining and the end of the process. A communication between GPU and RAM is more consuming than a communication between CPU and RAM. Therefore the identification is faster on SC

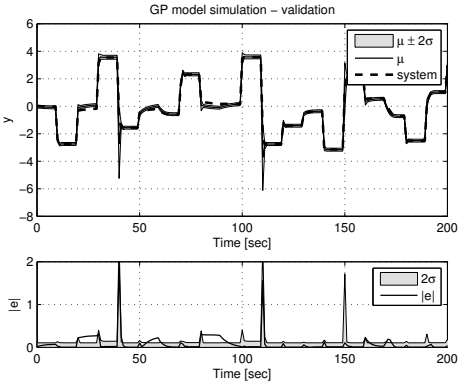

Fig. 5 Gaussian process model response on validation signal and comparison with the system response (top figure) and absolute value of model residuals with 95% confidence band (bottom figure) - the segment of first 200 samples

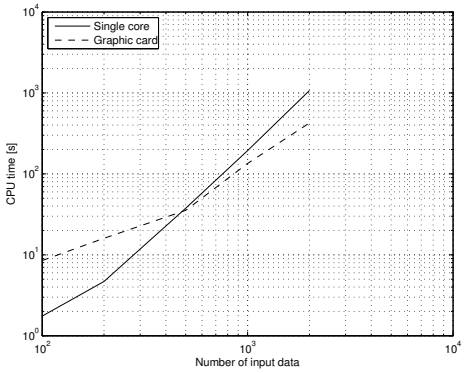

Fig. 6 Figure of computation times for the model identification versus input data dimension for different hardware configurations

computer until this overhead is majority of the whole computation time. Once the initial and communication overhead becomes sufficiently small part of the whole process, then a GPU turns out as more efficient solution.

The comparison of iterative dynamic model simulation is compared in two ways.

First, the computation times versus identification input data dimension for different hardware configurations were calculated. This comparison reveals acceleration of prediction with different sizes of the inverse of covariance matrix. Therefore the time complexity rises with a square. The results are depicted in Figure 7.

Similar to the identification also a simulation is faster on SC computer until the boundary of amount of data, where initial and communication overhead becomes sufficiently small part of the whole process, is reached. In our case this boundary is approximately 105 data which is quite lower than at the identification. After this boundary the simulation is likewise faster on the GPU and its factor rises similar to the identification.

The GPU in comparison to the SC computer is approximately 2.9 times faster for 500 data, for 1000 data approximately 4.0 times faster and for 2000 data approximately 5.3 times faster.

Second, the computation times versus validation input data dimension for different hardware configurations were calculated. This comparison reveals acceleration of simulation procedure with the fixed nonparametric dynamic system model. Therefore the time complexity rises linearly. The results are depicted in Figure 8.

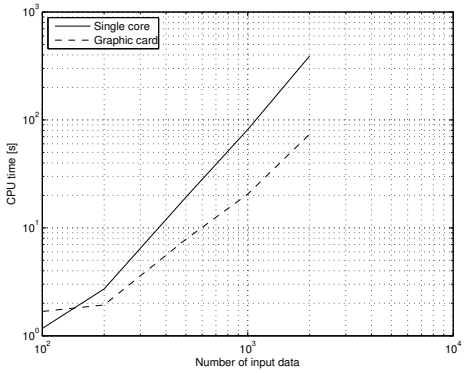

Fig. 7 Figure of computation times of the model simulation versus input data dimension for different hardware configurations

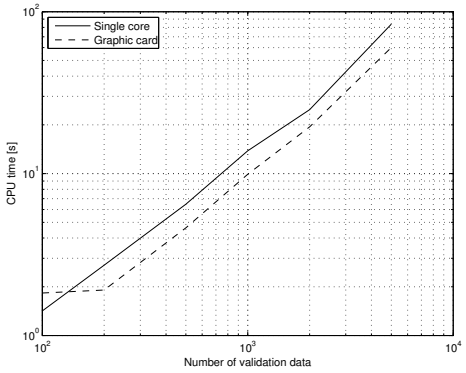

Fig. 8 Figure of computation time of the model simulation versus validation data dimension for different hardware configurations

Also for simulation with a fixed model holds that for small amount of data a computation is faster on a SC computer. Once the boundary is reached a computation on GPU becomes faster. In this case the boundary is approximately 80 data. Since a time complexity of this case has linear scale its factor of improvement is constant and is approximately 1.4. That means the computation on a GPU is 1.4 faster than on a SC computer for more than approximately 150 data.

5 Conclusions

The Gaussian process models are a flexible tool that can be used for dynamic system's identification. A drawback of the method is that its computational time rises with the third power of the number of input data used for model identification. This paper gives an assessment of different hardware configuration that can accelerate the computation.

For computations were used a single core personal computer and personal computer with graphic card. In the first case for a computation was used a CPU (only one core), while in the second case a GPU (all cores) were used.

From the case study it was obtained that a parallel computation is efficient for amount of data larger than the boundary when the initial and the communication overhead becomes sufficiently small part of the whole computation. Once this boundary is exceeded the parallelisation is reasonable. However, this parallel implementation of GP models on GPU is still work in progress, therefore, we believe there still exists room for improvements.

This study was performed from the user point of view to test the usability different computational platforms, therefore our future plans include implementation of the described algorithms on nowadays multi-core CPUs.

As hardware capabilities are improving constantly and research on efficient algorithms is on going the presented assessment might not be of a long term value. However, it offers the state-of-the-art comparison of affordable hardware configuration that might help to circumvent the computational issue in intermediate time before more efficient algorithms or better technology arise. With this it fulfills the purpose for which it was intended.

Acknowledgment

This work has been supported by the Slovenian Research Agency, grants Nos. P2-0001 and J2-2099.

6 References

[1] K. Ažman, J. Kocijan, Application of Gaussian processes for black-box modelling of biosystems. *ISA Transactions*, Vol. 46, No, 4, 443-457, 2007.

[2] J. Kocijan, A. Girard, B. Banko, R. Murray-Smith. Dynamic Systems Identification with Gaussian Processes, *Mathematical and Computer Modelling of Dynamic Systems*,Vol. 11, 411-424, 2005.

[3] J. Kocijan, K. Ažman and A. Grancharova. The Concept for Gaussian Process Model Based System Identification Toolbox. In *Proceedings of the InternationalConference on Computer Systems and Technologies* (CompSysTech), IIIA.23-1 - IIIA.23-6, Rousse, 2007.

[4] J. Kocijan, B. Likar. Gasliquid separator modelling and simulation with Gaussian-process models. *Simulation Modelling Practice and Theory*, Vol. 16 (8), 910-922, 2008

and
In *Proceedings of the 6th EUROSIM Congress on Modelling and Simulation* (EUROSIM 2007), 7 pages, Ljubljana, 2007.

[5] K.S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, Vol. 1(1), 4-27, 1990.

[6] J. Quionero-Candela and C. E. Rasmussen. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, Vol. 6, 1939-1959, 2005.

[7] C.E. Rasmussen and C.K.I. Williams. *Gaussian Processes for machine learning*. The MIT Press, Cambridge, MA, 2006.

[8] V. Volkov and J. Demmel. *LU, QR and Cholesky Factorizations using Vector Capabilities of GPUs*. EECS Department, University of California, Berkeley, 2008.

[9] M .harris *Parallel Prefix Sum (Scan) with CUDA*. 2008.

[10] *Cuda Programming Guide Version 2.3.1*. 2009.