Preprints of the 18th IFAC World Congress
Milano (Italy) August 28 - September 2, 2011

Optimal Control of the Process Systems Using Graphic Processing Unit

Arash Sadrieh * , **Parisa A. Bahri** **

* *School of Engineering and Energy, Murdoch University, Western Australia, 6150 (e-mail: A.Sadrieh@murdoch.edu.au)*
** *School of Engineering and Energy, Murdoch University, Western Australia, 6150 (Corresponding author , Phone: +618-9360-7227; E-mail: P.Bahri@murdoch.edu.au)*

Abstract:
In this paper the Graphic Processing Unit (GPU) is applied in order to improve the computational performance of process systems optimal control calculations. To apply GPU massive parallel architecture, a simplified version of interior point optimisation algorithm was selected and modified to fulfil special hardware requirements of GPU architecture. In this algorithm, a damped nonlinear Newton with preconditioned iterative linear solver was used to solve Dual-Primal equations. The comparison of results between this implementation and standard CPU-based implementation shows considerable improvement in computational performance.

Keywords: GPGPU; Optimal Control; Interior Point; Nonlinear Programming.

1. INTRODUCTION

Over the past decade, creating optimal control policies for process systems has received considerable attention from both academia and industries. Numerous optimisation algorithms have been developed and applied in software tools for modelling, simulation and optimisation of process tools. Among these tools, *equation oriented* tools such as Aspen Custom Modeller (ACM, Aspen Technology, Inc) and gPROMS (PSE, Inc) played a significant role as they simplifying modelling and optimisation tasks for the process modellers.

In order to build an efficient and robust optimal control policy, a process modeller should perform complex and time consuming tasks that usually include various adjustments in the model. Running optimisation algorithm of course is an important part of these tasks and due to computationally intensive nature of optimisation algorithms considerable amount of time is used while the process modeller is waiting for the optimisation algorithm results. In the development of optimal control policy, the effects of computational delay, therefore, is an important consideration, specifically in dealing with the models that contain nonlinear mathematical models with many equations and variables.

In this study, the specific architecture of Graphic Processing Unit (GPU) is exploited as a new hardware co-processor to solve Process Systems Optimal Control (PSOC) problems. The application of massively parallel architecture of GPUs resulted in improvement of computational performance of classical nonlinear interior dual-point optimisation algorithm.

The structure of this paper is organised as follows: Firstly, in section 2 general background information is provided on General Purpose GPU (GPGPU). The interior point algorithm is then explained in section 3 as an algorithm that can satisfy different requirements raised from optimal control theory in the process systems. Subsequently, implementation details of different parts of the algorithm is explained in section 4 and finally in section 5 the results obtained are compared against standard CPU-based implementation of the algorithm.

2. BACKGROUND

The GPU, a component found in most new PCs, is employed traditionally for 3D computer graphics and is hosted in graphical adaptor. This processor chip is designed to facilitate the execution of very high number of threads in parallel. This means, a similar piece of software, is executed independently on very large amount of data. To put the speed of GPUs into context, take NVIDIA Fermi architecture introduced in 2009 (Wasson [2009]). This processor has 512 computing cores and can sustain a peek rate of more than 500 Giga FLoating points Operations Per Second (GFLOPS [1]), compared to the fastest CPU at the time which operated a peak of less than 20 GFLOPS .

The idea of using GPUs for general computation has only recently gained attention with the definition of the GPGPU given in 2002 by Harris [2003]. After that many successful applications were reported in the literature claim to achieve speed ups in orders of magnitude compared to optimised CPU implementations (see Owens et al. [2007]).

From a programmers point of view, the GPU can be assumed as a co-processor that cooperates with a main processor (i.e. CPU). Host processor (CPU) and device

[1] GFLOPS is a computing performance measurement unit that is mainly applied in scientific computing applications.

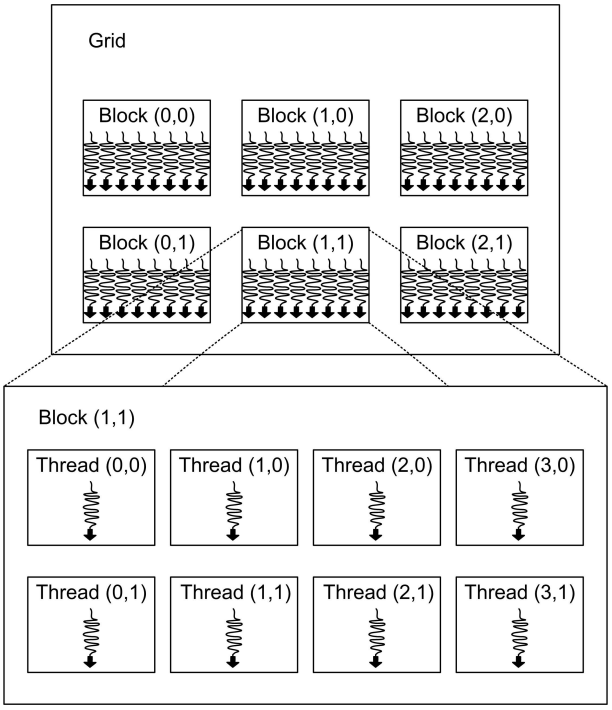

Fig. 1. A Simple Grid: adapted from Nvidia [2007] and shows hierarchical structure of a grid that includes 6 kernel blocks where each block is including 8 threads.

processor (GPU) manage their own memories and data is transferred from host memory into device memory and vice versa. Data parallel algorithms are implemented on the device by providing a function algorithm that should be executed many times on different data. The functions that are specifically developed for the device are called kernels. Figure 1 shows a simple example for the execution model of the kernels. In this model, a grid contains a batch of similar kernel blocks that run concurrently on the device. In a grid, a kernel block describes a group of kernel threads that can cooperate with each other efficiently through the fast shared memories available on the GPU architecture. For more detailed information see Nvidia [2007]. Kernels are implemented using different programming environments such as Compute Unified Device Architecture (CUDA) and OpenCL.

On the other hand, the application of parallel processing methods in PSOC problems has a history of more than three decades (Larson and Tse [1973]). Benner et al. [2008] used parallel processing techniques to solve algebraic Riccati Equations raised from linear-quadratic optimal control problems.

Biegler et al. [2002] explained that the simultaneous collocation approach for PSOC problems can be implemented on parallel processing architectures.

While significant progress has been made in applying parallel processing algorithms in the solution of PSOC problems, existing approaches should be executed on very expensive parallel architectures. Moreover, these parallel architectures normally are very hard to access for the process modellers. Introducing GPU chips tackle the price and accessibility challenge of parallel architectures. This provides great potential for optimal control solvers to harness GPU chips in order to improve the computational performance of the solution. Schenk et al. [2008], consequently developed a GPU-based direct linear solver and integrated it with Interior Point OPTimize (IPOPT) optimisation algorithm.

Considering the fact that PCI-E bus is used for data transfer between host and device and the fact that the maximum theoretical transfer speed in current PCI-E bus is 8 GBPS, it can be concluded that in the previous GPU-based approaches host/device data exchange could become a very important performance bottleneck. To overcome this bottleneck, it is necessary to reduce the host-device data transfer (see Nvidia [2007]). The approach explained in this paper, therefore, addresses this issue by providing GPU-based parallel implementation for different parts of an optimal control solver algorithm.

3. OPTIMAL CONTROL

In the equation oriented tools, to solve the PSOC problem, the process is modelled dynamically by differential-algebraic equations (DAE). Differential equations typically, are produced from energy, mass and momentum balances while phenomenological constitutive equations and process constraints are the main sources for the algebraic equations.

After developing the process model, a set of control and state variables are selected for optimisation and the cost function is specified based on these variables. Subsequently, steady state and dynamic constraints and final termination time are determined and then the software applies a numerical algorithm to solve the PSOC problem. To explain the approach taken by the software tool, consider the cost function J of a nonlinear optimal control problem that is expressed by:

$$\min_{u(t)t\in[0,t_f]} J = \phi(z(t_f),t_f) + \int_0^{t_f} L(z(t),u(t))\,\mathrm{d}t \tag{1a}$$

$$s.t.: \frac{dz}{dt} = f_0(z(t),u(t),t) \tag{1b}$$

$$g(z(t),u(t),t) = 0 \tag{1c}$$

$$u^L(t) \le u(t) \le u^U(t) \tag{1d}$$

$$z^L(t) \le z(t) \le z^U(t) \tag{1e}$$

Where $z(t)$ and $u(t)$ are state and control variables at time t, and these variables are bounded with lower and upper bound limits (i.e. $u^L(t)$,$u^U(t)$,$z^L(t)$,$z^U(t)$). In equation 1, t_f denotes terminal time and ϕ and L are endpoint cost and system Lagrangian, respectively. In equation oriented tools, normally *simultaneous collocation* (Biegler et al. [2002]) approach is applied in order to solve the optimisation problem expressed by equation (1). In this approach, the control variable and bound limitations are discretized into N dynamic stages over time. Simultaneously, at each dynamic stage the state variable profile is also discretized into M parts that convert equation (1) into a Nonlinear Programming (NLP) problem given by:

$$\min_{u\in R^N} J^* = \phi(z_{M,N}, t_f) + \sum_{k=1}^{N}\sum_{i=1}^{M} L(z_{i,k}, u_k)\,\Delta t \tag{2a}$$

$$s.t.: \frac{dz}{dt_{i,k}} = f_0(z_{i,k}, u_k) \tag{2b}$$

$$g(z_{i,k}, u_k) = 0 \tag{2c}$$

$$u_k^L \le u_k \le u_k^U \tag{2d}$$

$$z_k^L \le z_{i,k} \le z_k^U \tag{2e}$$

$$k = 1, \ldots, N, i = 1, \ldots, M \tag{2f}$$

Where $z_{i,k}$ denotes the value of state vector at the time $t_{i,k} = (i + M\,k)\,\Delta t$ and u_k is the value of manipulated variables at stage k when $t_k = M\,k\,\Delta t$ and Δt represents a fixed time step for the system that is equal to $\frac{t_f}{N\,M}$. Problem (2) is solved with Primal-Dual Nonlinear numerical algorithm.

3.1 Primal-Dual Nonlinear Interior Algorithm

In the PSOC problems, problem (2) becomes a very large NLP optimisation problem. Although a wide range of open source and commercial optimisation packages offer methods to solve this problem, in this study, primal-dual interior-point algorithm with a basic line-search method is selected (Wächter and Biegler [2006]). The main reasons for this selection are: 1) Different parts of the algorithm can be implemented on parallel architecture. 2) The algorithm is well documented and an open source implementation is available. 3) It can solve nonlinear optimal control problems. 4) It has the ability to handle problems with a large number of inequality constraints that are normally raised from optimal control problems of process system. 5) In this algorithm, the problems with high ratio of state variables to discretized control variables can be optimised. This is a critical feature for process optimal control problems where the number of state variables is often thousand times larger than the number of discretized control variables.

To simplify the notation, we can express problem (2) by:

$$\min_x J^*(x) \tag{3a}$$

$$s.t.: c(x) = 0 \tag{3b}$$

$$x \ge 0 \tag{3c}$$

where $x : (\frac{dz}{dt_{i,k}}, z_{i,k}, u_k, t) \in \Re^n$ and $J^* : \Re^n \to \Re$ and the vector $c : \Re^n \to \Re^m$ denotes the constraints expressed in (2b) and (2c). NLP problem (3) is reformulated to an associated barrier problem as:

$$\min_x \varphi_\mu = J^* - \mu \sum_{i=1}^{n} ln(x_i) \tag{4a}$$

$$s.t.: c(x) = 0 \tag{4b}$$

In formulation (4), μ denotes barrier parameter that was applied in the optimisation algorithm to calculate the optimum solution. In this algorithm, μ is gradually reduced until the convergence to the optimal solution is achieved. This is similar to using the homotopy method (see Watson and Haftka [1989]) for Primal-Dual equations expressed by:

Algorithm 1. Primal-Dual Nonlinear Interior Algorithm Adapted from Wächter and Biegler [2006]

```
Initialize.
k ← 0
j ← 0
while overall problem is not converged do
    while barrier problem is not converged do
        μ_{j+1} ← reduce(μ_j)
        j ← j + 1
        if K ≠ 0 then
            Break the inner while loop.
        end if
    end while
    Compute the search direction. ▷ Newton method.
    Backtrack step length.
    if step length is too small then
        Run feasibility restoration phase.
    end if
    Accept the trial point.
    k ← k + 1
end while
```

$$\nabla J^*(x) + \nabla c(x)\lambda - z' = 0 \tag{5a}$$

$$c(x) = 0 \tag{5b}$$

$$XZ'e - \mu e = 0 \tag{5c}$$

Where $z' \in \Re^n$ and $\lambda \in \Re^m$ denote the Lagrange multipliers associated with (3b),(3c) and Z', e, X are defined by:

$$Z' = diag\,\{z'_1, z'_2, \ldots, z'_n\} \tag{6a}$$

$$X = diag\,\{x_1, x_2, \ldots, x_n\} \tag{6b}$$

$$e = (1, 1, \ldots, 1)^T \tag{6c}$$

A simplified version of the nonlinear interior point optimisation method (Wächter and Biegler [2006]) is provided in algorithm (1). This algorithm iteratively solves optimisation problem (2) by solving a sequence of nonlinear equations (5). Newton method is applied to find the search direction (i.e. the solution of nonlinear equation (5)). To ensure the algorithm achieves global convergence, a back track line search is performed towards the calculated search direction values. The back tracking routine searches for a step length that sufficiently improves the objective function or improves the constraint violations. The algorithm starts restoration phase if the step length is smaller than a certain value. During this phase, another trial point that satisfies the constraints is selected and the algorithm continues the search from this new trial point.

4. IMPLEMENTATION

In order to improve the computational performance of PSOC solution, a hybrid GPU/CPU approach is taken. The main idea is firstly to exploit GPU hardware architecture in specific optimisation problem raised from the structure of problem (2) and secondly minimize data transfer between host and GPU. To implement the GPU kernels, algorithm (1) is divided into components in the way that each component performs a specific task. These components are model evaluator, convergence tester, nonlinear solver, line searcher and restoration mechanism.

Implementation details of these components are provided in the following sections. The algorithm is implemented on GPU through the NVIDIAs CUDA framework (v. 3.2) based on Fermi architecture.

4.1 Integration with the Simulation Tool

Generally, PSOC problems are formulated in equation oriented tools and due to the complex nature of these problems, reformulating them to the specific format (as required in our algorithm) is very time consuming and error prone. In this implementation, therefore, some utilities were created to automatically perform this task. These utilities contain two subsystems called Code Generator and Optimisation Solver Interface. While current implementation is based on ACM equation oriented tool, the concept is quite similar in other equation oriented tools.

The code generator receives the process model in C programming language format (i.e. previously exported from the ACM model) and then automatically analyses and categorizes the model. The results are subsequently applied to build equivalent CUDA kernel for the model. Moreover, the equation groups with similar structure are merged into a single kernel and by applying method of Automatic Differentiation, GPU-based evaluator of gradient and Hessian is also generated. The model evaluators component should be generated just when the structure of model changes.

The Optimisation Solver Interface is a specific interface component that is developed based on Aspen Open Solver interface. This interface enables the integration of our implementation into ACM equation oriented environment. This allows the process modeller to use our optimisation solver directly from the ACM user interface. The integration has some advantages for the process modellers: Firstly, it helps process modellers applying GPU computational benefits without any understanding about GPU related programming complexities and secondly, giving an opportunity of building upon the existing models so as to not lose investment in the legacy models.

4.2 Model evaluator

As mentioned previously, the model evaluator component is generated in the code generator subsystem and is responsible for evaluating objective function J^* , gradient of objective function ∇J^* , constraint function c, gradient of constraint function ∇c and Hessian matrix of the Lagrangian function (i.e. $\nabla^2_{xx}(\nabla J^* + c^T \lambda - z')$). Variables J^*,∇J^*, ∇c and ∇c are applied directly in nonlinear equations (5) while Newton nonlinear solver uses the Hessian matrix during the numerical solution of equations (5). Moreover, model evaluator results are often applied in other parts of the algorithm such as line search and convergence test. The component assumes the input values are located on device memory but it has the ability to provide the results in either device or host memory. This component contains a separate kernel for each group of similar equations and the kernels are lunched simultaneously in order to evaluate the model. The finite element applied in simultaneous collocation approach resulted in generation of specific GPU-base model evaluators. In this evaluator parallel computing requirements of GPU architecture is satisfied. The parallel requirement for the algorithm is that it should execute a similar piece of code (i.e. equations) on a very large amount of data (such as discretized control and state variables). Consequently, the model evaluators of problem (2) can achieve very high computational performance on the GPU architecture. In theory, this means that the algorithm is completed after $O(\frac{n}{p})$ operations where p is the number of parallel cores in the device.

It should be noted that due to the structure of PSOC problem the Hessian matrix is very large and sparse and in this implementation GPU-based sparse matrix kernels are applied for general sparse operations such as matrix vector multiplication and dense/sparse multiplication.The algorithm takes an approximation approach when Hessian matrix is not available, applying a simple GPU-based inexact quasi-Newton method.

4.3 Convergence Tester

The error for the results obtained from algorithm (1) cannot be calculated exactly due to the unknown solution of PSOC problem. Different heuristic methods, however, are applied in order to testify whether the algorithm should be terminated or the iteration to be continued. In the standard implementation constraint (7) is applied on residuals *fixed vector norm* value to decide on overall problem convergence.

$$\max\{\alpha_1\|R_1\|, \|R_2\|, \alpha_2\|R_3\|\} \leq \epsilon_{tol} \tag{7}$$

Where R_1, R_2 and R_3 respectively denote residual vectors of equations (5a), (5b), (5c). These residuals are calculated in the model evaluator component and scalar values α_1 and α_2 are used to scale norms in the scenarios that complexity handling of large numbers is required. The constant tolerance value ϵ_{tol} is provided by the process modeller.

Equation 7 implies that the convergence tester component should find the element with the maximum value in all of the three vectors $\alpha_1\|R_1\|, \|R_2\|, \alpha_2\|R_3\|$ and compare the maximum against the tolerance value.

To build a CUDA kernel for this component, the parallel reduction method proposed by Harris [2008] was used. This method suggests that every thread reads two (or more) elements from the input data buffer (i.e. all the elements of $\alpha_1 R_1$, R_2 and $\alpha_2 R_3$) and then they should perform reduction operator on these values (in our specific case, finding the maximum). Afterwards, the computed results should be stored in the location of the first element. Considering multiple kernels running in parallel, after each operation, the input data size is reduced into half (or less). The operation is called reduction pass and kernels carry out reduction passes until input data reduces to one element vector. Parallel reduction algorithm needs $O(\log n)$ passes to complete, in contrast to the equivalent sequential algorithm which requires $O(n)$ operation.

4.4 Newton Nonlinear Solver

Newton nonlinear solvers are a class of numerical algorithms, applied to find the solution of nonlinear equa-

tions. In our implementation, a specific GPU-based Newton algorithm was applied to find the numerical solution of equations (5) where the results from nonlinear solver $(d^x, d^\lambda, d^{z'})$ are called search direction. At each iteration of this nonlinear solver, the system of nonlinear equations (5) is approximated by the matrix of linear equations (8). The nonlinear solver starts with initializing $d^x_{k'}$, $d^\lambda_{k'}$, $d^{z'}_{k'}$ with an initial guess for the solution and then iteratively calculates the corrections for the solution vector using equations (8). More specifically, at the step k' the solution for (8) is the k'th Newton step size.

$$\begin{bmatrix} W_{k'} & A^T_{k'} & -I \\ A_{k'} & 0 & 0 \\ Z'_{k'} & 0 & X_{k'} \end{bmatrix} \begin{pmatrix} d^x_{k'} \\ d^\lambda_{k'} \\ d^{z'}_{k'} \end{pmatrix} = - \begin{pmatrix} \nabla J^*(x_{k'}) + A_{k'}\lambda - z'_{k'} \\ c(x_{k'}) \\ XZ'e - \mu_j e \end{pmatrix} \tag{8}$$

Here A_k is the gradient of the constraint function (∇c) and W'_k is the Hessian matrix. These matrices are calculated in the model evaluator component.

In the nonlinear Newton iterations, equation 8 is solved by undertaking two steps: Firstly symmetric sparse linear equation (9) is solved for $(d^x_{k'}, d^\lambda_{k'})$:

$$\begin{bmatrix} (W_{k'} + X^{-1}Z'_k) & A_k \\ A^T_{k'} & 0 \end{bmatrix} \begin{pmatrix} d^x_{k'} \\ d^\lambda_{k'} \end{pmatrix} = - \begin{pmatrix} S(J^*, A, x_{k'}, z'_{k'}) \\ c(x_{k'}) \end{pmatrix} \tag{9}$$

Where S denotes a function that is produced from elimination of $d^{z'}_{k'}$ row in equation (8). To implement this step on a GPU a preconditioned conjugate gradient solver (Buatois et al. [2007]) was used. Conjugate gradient is a method, applied to solve linear equations indirectly. This method applies a preconditioner matrix to improve the solution efficiency. The preconditioner is created based on the method proposed by Bergamaschi et al. [2004].

The next step in solution of (8) is solving equation (10) for $d^{z'}_{k'}$:

$$d^{z'}_{k'} = \mu_j X^{-1} e - z'_{k'} - X^{-1}_{k'} Z_{k'} d^x_{k'} \tag{10}$$

A CUDA kernel was implemented that calculates the right hand side of equation (10) for all the elements of vector $d^{z'}_{k'}$. Other parts of Newton nonlinear solver is implemented on the device by applying GPU-based standard matrix/vector operations library. From computational point of view, the important factor about these parts is that in the whole operation of nonlinear solver, it is not necessary to access any information on the host memory. That means, the host/device data transfer overhead is completely eliminated.

4.5 Line Search

Backtracking line search method is applied to ensure that the optimisation algorithm (1) can converge from almost any initial guess for solutions. This method adjusts the step size in the search direction to sufficiently improve objective function or constraint violation.

Due to hardware structure of GPU, simultaneous evaluation of a model, for several variables, has a similar computational time of a single variable. Assume Q is the number of concurrent model evaluators on GPU where their evaluation time is roughly equal to a single evaluation. In PSOC problems, normally Q is between 50 to 150 and the exact value of Q is determined based on the model structure and hardware. The GPU-based line search algorithm receives constant $\beta \in (0, \frac{1}{2}]$ and d^x_k as input parameters and is started by running a group of Q threads in parallel. In this group, thread m is responsible for building x'_m and then evaluating model objective function and constraint functions based on x'_m. The variable x'_m is defined by:

$$\begin{aligned} &1 \le m \le Q \\ &\alpha_m = \beta^m \\ &x'_m = x_k + \alpha_m \, \mathrm{d}^x_k \end{aligned} \tag{11}$$

When the operation of this group is completed the optimum α_m is selected as a step size. Considering the best case and worse case analyses, it can be concluded that on average, this component is performing theoretically $\frac{Q}{2}$ faster compared to equivalent sequential line search.

4.6 Restoration mechanism

The restoration mechanism component is applied when the line search step size becomes too small or in other special cases when the progress to the solution is becoming too difficult. From a computational point of view, this component is not very important due to the fact that it called in special cases. Consequently in this implementation whenever the main algorithm detects that the restoration is necessary the CPU-based standard is called. It should be noted that the data transfer overhead introduced here is negligible compared to the time of optimisation algorithm iteration.

5. NUMERICAL RESULTS

Optimal control problem of a crystallizer model, explained by Pantelides and Oh [1996] and incorporated as an ACM example, is selected to illustrate the proposed approach. The model describes a continuous mixed suspension mixed product removal (CMSMPR) crystallization unit. In this model, potassium sulphate crystals are produced from aqueous solutions. It was assumed that crystal breakage is not important. To simulate the model in dynamic mode, it was supposed that at start-up, the crystallizer was filled with pure water. More specifically, initial liquid concentration of solute and logarithmic number density for all non zero crystal sizes are set to zero. This model is selected as it has different types of equations including mass, energy balances (ODE), crystal population balance (PDE), magma density (integral form), and several algebraic equations.

The objective function is maximizing yield during the start up of crystallization by controlling the temperature T over a time horizon of $t_f = 5$ hours. Considering yield as a positive value, the optimal control problem is therefore formulated as:

$$\min \int_0^{t_f} (-Y) \, \mathrm{d}t \tag{12}$$

Subject to dynamic DAE model implemented in ACM and,

$$25^\circ C \le T \le 50^\circ C \tag{13}$$

Where Y denotes percentage yield in the CMSPMPR unit. All the tests are performed on a PC with Intel 2.98 GHz Core (TM) 2 Duo Processor E7500 with NVIDIA Fermi GPU (GTX480).

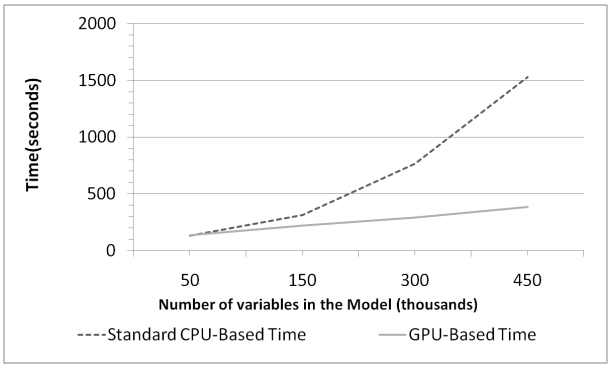

Fig. 2. Performance Comparison: The current approach is compared with standard CPU-based implementation while CMSMPR problem is solved for different number of variables.

5.1 Discussion

To confirm the correctness of the results, the results obtained from GPU-base implementation were compared to that calculated by standard IPOPT algorithm and they were confirmed to be the same.

In this specific example, when number of variables is about 450,000, the computation time for a single model evaluation is roughly equal to the time of 119 concurrent model evaluations. That means for CMSMPR case study we have $Q = 119$, more specifically, the GPU processor is saturated in 119 concurrent model evaluations and can not perform more than 119 evaluations at the same time.

To measure the scalability of the approach, the number of equations, variables and constraints in the PSOC were changed and for every model size PSOC problem is optimised. The model size was adjusted by discretization spacing preference in the DAE model, discretization of temperature over time horizon (i.e. control variable) and discretization of state variables over time. Changing these parameters resulted in different number of variables for the model. As it is shown in figure (2) current approach is outperforming standard CPU-based implementation particularly when the number of variables is increased. For example, during optimising a PSOC containing more than 450,000 variables and 1,200 degrees of freedom, the current approach runs about 4.0 times faster compared to the standard approach.

6. CONCLUSIONS

The results show a considerable improvement in the computational performance. It should be noted that the maximum number of variables and equations in the PSOC problem is directly limited to maximum memory available on the single GPU cards (i.e. between 2GB to 6GB). This limitation implies that a single GPU can not be used for a model containing, roughly, more than 50,000 equations. However, we could expect that this limitation will be addressed in future by introducing more memory on the cards. Considering the results and the growth rate in the computational power, price and the availability of GPU chips, it can be concluded that these chips can be considered as very attractive co-processor in equation oriented software tools. Further work will involve optimising kernels codes and applying similar method in model predictive control of process systems.

ACKNOWLEDGEMENTS

The authors would like to thank NVIDIA and NVIDIA Professor Partnership Program for their hardware donations.

REFERENCES

P. Benner, E.S. Quintana-Ortí, and G. Quintana-Ortí. Solving linear-quadratic optimal control problems on parallel computers. *Optimization Methods and Software*, 23(6):879–909, 2008.

L. Bergamaschi, J. Gondzio, and G. Zilli. Preconditioning indefinite systems in interior point methods for optimization. *Computational Optimization and Applications*, 28(2):149–171, 2004.

L.T. Biegler, A.M. Cervantes, and A. Wächter. Advances in simultaneous strategies for dynamic process optimization. *Chemical Engineering Science*, 57(4):575–593, 2002.

L. Buatois, G. Caumon, and B. Lévy. Concurrent number cruncher: An efficient sparse linear solver on the gpu. In *High Performance Computing and Communications*, volume 4782 of *Lecture Notes in Computer Science*, pages 358–371. Springer Berlin / Heidelberg, 2007.

M.J. Harris. *Real-time cloud simulation and rendering*. PhD thesis, Citeseer, 2003.

M.J. Harris. Optimizing parallel reduction in CUDA. 2008. URL `http://www.mendeley.com/research/optimizing-parallel-reduction-cuda/`.

R.E. Larson and E. Tse. Parallel processing algorithms for the optimal control of nonlinear dynamic systems. *Computers, IEEE Transactions on*, C-22(8):777 –786, aug. 1973.

C. Nvidia. Compute Unified Device Architecture Programming Guide. *NVIDIA: Santa Clara, CA*, 2007.

J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A.E. Lefohn, and T.J. Purcell. A Survey of General-Purpose Computation on Graphics Hardware. In *Computer graphics forum*, volume 26, pages 80–113. Wiley Online Library, 2007.

C.C. Pantelides and M. Oh. Process modelling tools and their application to particulate processes. *Powder Technology*, 87(1):13–20, 1996.

O. Schenk, M. Christen, and H. Burkhart. Algorithmic performance studies on graphics processing units. *Journal of Parallel and Distributed Computing*, 68(10):1360–1369, 2008.

A. Wächter and L.T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

S. Wasson. Nvidia's Fermi gpu architecture revealed, 2009.

L.T. Watson and R.T. Haftka. Modern homotopy methods in optimization. *Computer Methods in Applied Mechanics and Engineering*, 74(3):289–305, 1989.