Tutorial Paper: Parallel Architectures for Model Predictive Control

George A. Constantinides

Abstract— **This tutorial paper surveys recent developments in parallel computer architecture, focusing on the field-programmable gate array and the graphics processor. We aim to illustrate the potential of these architectures for the type of high-speed numerical computation required in on-line optimization for model predictive control. While significant performance advantages can be gained by migrating existing control algorithms to these processor architectures, in order to realise their full potential, further research is needed at the boundary of control theory, digital electronics, and computer architecture. We survey some of the open questions in this area.**

I. INTRODUCTION

This tutorial paper introduces some recent developments in the field of computer architecture, relating to parallel processing, and describes the significant potential impact on the area of model predictive control (MPC). In particular, we will survey the potential of domain specific architectures such as graphics processors (GPUs) and field-programmable gate arrays (FPGAs) to accelerate the computations required in the on-line optimization at the heart of MPC, opening up new applications for MPC in areas where the computational load has been considered too great. As an example of the potential, we will report on recent advances made in the high-speed solution of systems of linear equations using the conjugate gradient method implemented with an FPGA [14].

Our focus is on using the FPGA to implement a computer architecture that is customised to the type of computation required, through the development of a customised data processing engine, a customised memory subsystem, and customised number representations. After introducing the optimization problem required by MPC, we focus on the main computational bottleneck: the fast solution of a large number of systems of linear equations, resulting from computing the search direction in an interior point method [11]. Some details are presented of a customised architecture for solving such systems [14], and comparisons are drawn to a general purpose microprocessor; we demonstrate that the FPGA-based engine is able to out-perform both the measured and the peak theoretical capability of the microprocessor.

When customising a computational architecture to the MPC applications, several important research questions arise, regarding the most efficient use of the silicon, in order to get the maximum performance benefit. The potential of FPGA architectures to allow the customisation of number representations [20], and the performance gap between high and low precision operation in GPUs [3] means that there is an inherent tradeoff between numerical accuracy and performance in these architectures. We shall consider this problem in further detail, and highlight some of the open interdisciplinary research questions that need to be considered by both the control theory community and the digital electronics community, in order to make the most of these new developments.

This work was supported by the Engineering and Physical Sciences Research Council (UK) under grant number EP/C549481/1.

George A. Constantinides is with the Department of Electrical and Electronic Engineering, Imperial College London, Exhibition Road, London SW7 2AZ, U.K. george.constantinides@ieee.org

II. CUSTOM COMPUTER ARCHITECTURES

The landscape of computer architecture is changing rapidly at present, and new opportunities for research are coming to the fore [1]. To a large extent, the reason for this change is one with which any computer user will be familar: clock frequency is no longer scaling significantly with time. However, clock frequency is not the only driver of computational performance; the degree of parallelism provides the other driver, hence the move to two and now four processor cores on desktop computers. In parallel with the advance of mainstream computer architectures, there have been two developments emerging over the past ten years in non-standard, *domain specific* architectures: the graphics processor (GPU) and the field-programmable gate array (FPGA).

Graphics processors were once very specialised devices, capable only of performing the shading and rendering required by low-level graphics routines. As the games industry has evolved, incorporating more physical simulation into gaming engines, so the graphics processor has evolved to be a powerful general floating-point compute engine in its own right. Today's GPUs, such as the nVidia GeForce GTX 280, include hundreds of processor cores, and a sophisticated memory subsystem [2]. For the consumer-oriented price, the theoretical peak single precision floating point performance of these processors is unrivalled, at nearly 1 TFLOP/s. The domain specificity of the GPU remains, however, in two main ways. Firstly, the performance of the GPU on double precision floating point is 2 to 10 times worse [3], partly because at present the games industry simply does not require high performance support for double precision computation. Secondly, the performance of the GPU can only achieve close to its theoretical peak if there is sufficient parallelism in the application, some thousands of parallel threads, and the memory storage can be arranged appropriately.

A radically different form of computational architecture is presented by the FPGA. An FPGA is a two dimensional array

of simple hardware units, such as small programmable read-only memories (ROMs), larger random-access memories (RAMs), and some arithmetic units, together with a flexible routing structure, as depicted in Fig. 1. The content of each of the memories, together with the information defining the topology of the interconnection between the units, can be programmed by uploading a bitstream to the device. As a result, arbitrary circuits can be implemented directly within the underlying general purpose programmable fabric. The potential to use these devices for high-performance computation arises because the computer architecture can therefore be customised to whichever task needs to be performed at a given instant. Such specialisation of the architecture has the potential to lead to signficant performance advantages. For example, while a general purpose architecture like a Pentium needs to implement any algorithm, the FPGA could be programmed to implement a computational engine solely suited for solving 10×10 systems of linear equations. While a GPU needs to support single-precision floating point computation, an FPGA-based architecture can be specialised to support whatever precision is required by the underlying application. This concept of creating special purpose computational hardware is not new - in some sense it harks back to the form of computer architecture employed by Turing and his co-workers during the 1940s [4], and again by the upsurge in work on systolic arrays during the 1980s [5]. However, the increasing fabrication costs, together with the design effort required, mean that application-specific circuits (ASICs) can generally only be supported in the mass markets such as mobile telephony and computer gaming. The power of the FPGA is that it amortizes these costs over a large number of customer designs, enabling specialised circuits to be implemented within a general purpose underlying fabric. A small but active community of researchers has been investigating FPGA-based computation since the early 1990s, but it is only recently that such work has received main-stream attention, due to two factors: FPGA logic density has now increased to the extent that it is feasible to perform hundreds to thousands of operations in parallel, and mainstream computer architectures have stopped scaling in clock frequency.

The price one pays for the flexibility provided by an FPGA is an overhead in the achievable clock rate. Thus, while mainstream microprocessors currently operate in the region of 2 GHz, even high-speed FPGA designs are unlikely to run faster than 300 MHz. It follows directly that a necessary condition for FPGA acceleration is the existence of sufficient parallelism in the target application to overcome this gap - a factor of approximately $7\times$. Many applications, especially from traditional high-performance computing, but increasingly from embedded systems such as control, easily offer potential parallelism above this factor, resulting in significant speedups compared to state-of-the-art microprocessors. However, the speedups achieved can vary significantly, depending on the affinity of the algorithm for the FPGA architecture. Particularly impressive acceleration, of two to three orders of magnitude, have been achieved for string-matching algorithms such as genome sequencing [6].

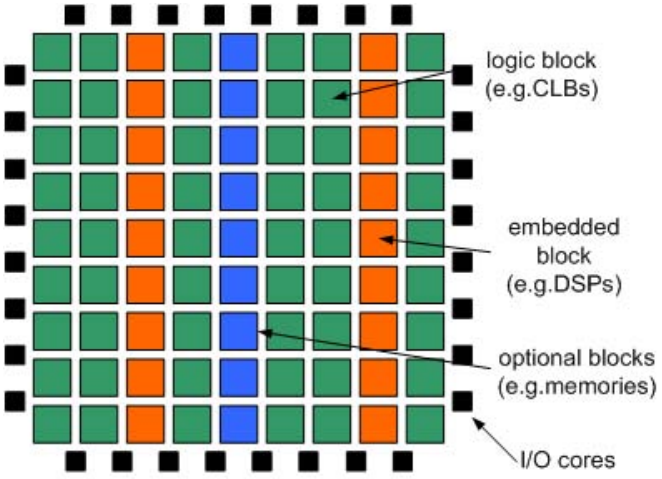

Fig. 1. An example FPGA device. Combinational Logic Blocks (CLBs) consist of small programmable ROMs, memories are larger RAM blocks, in the region of 1k bit, and DSP blocks are dedicated multiply/add circuitry. The programmable routing fabric is not shown.

For numerical computation, between $10\times$ [7] and $80\times$ [8] speedups have been achieved, depending on the accuracy requirements and the finite precision number representation used.

III. COMPUTATIONAL BOTTLNECKS IN MPC

Model predictive control (MPC) is the most commonly implemented advanced control technique in the process industries today [9]. The fundamental idea behind MPC is straightforward: at sample period t an optimal control problem, which uses a model of the system to be controlled, is solved with the aid of constrained numerical optimization methods. Following this, only the first part of the solution is implemented for the duration of the sample period h. Due to model uncertainty and disturbances, the actual output trajectory may deviate from the predicted trajectory, thus a measurement of the actual output at the next sample instant $t + h$ is taken, and the optimal control problem is updated with the new measurement. This process of measuring, solving a constrained optimization problem and implementing only the first part of the optimal control sequence is repeated at future sample instants.

From an algorithmic perspective, most MPC implementations result in the requirement to solve, at each sample instant, a quadratic program (QP) of the form

$$\begin{aligned} &\min_\theta \ \theta^T H \theta + \theta^T G x \\ &\text{subject to} \ \ A\theta \leq m + Cx, \ D\theta = e + Fx \end{aligned} \tag{1}$$

where θ is a vector of future states and inputs, x is a measurement or estimate of the state of the system at the current sample instant, and the matrices (H, G, A, C, D, F) and vectors (m, e) are functions of the model dynamics, constraints and weights in the cost function.

It is clear that the range of applications to which MPC can be applied is heavily dependent on the ability to solve

constrained nonlinear optimization problems, such as the above QP, reliably and efficiently in real-time.

There are two main approaches to real-time solution under a tight sample time constraint. One can either solve the optimization problem in a parametric fashion, off-line, and then in real-time perform a fast table lookup [10], or one can try to solve the optimization problem on-line at each sample instant. The parametric approach is suitable for some problems, but in the general case the complexity of the computation time and memory requirements of the solution grow exponentially with the problem size. By solving the optimization problem on-line with interior point methods [11], we can guarantee polynomial growth in solution time with problem size. These two techniques are therefore both attractive solutions, depending on the scale and the structure of the optimization problems; for the remainder of this paper, we shall consider the general on-line technique.

If interior point methods are used to solve the optimization problem (1), then the largest computational bottleneck arises from the need to repeatedly solve systems of linear equations of the form $Qy = r$ in order to determine a search direction. These systems may be small and dense, with symmetric positive definite Q, if the equality constraints in (1) are first eliminated, or larger but band structured with symmetric but potentially indefinite Q otherwise [12]. One of the central problems for customised hardware is therefore to determine what potential computational throughput can be achieved for such problems.

IV. HIGH-PERFORMANCE LINEAR ALGEBRA

There has been some recent research into high-performance FPGA-based linear algebra [13], [14], [7]. In this section, we will present one example, an FPGA implementation of the Conjugate Gradient algorithm [15], an iterative method for solving positive definite systems of linear equations. Both iterative and direct methods for solving systems of linear equations have shown some promise in FPGA-based computation [16], [14], [7], but we focus here on an iterative method for two reasons, one technology-specific, and one technology-independent. Firstly, modern iterative methods are likely to map particularly well to architectures where the multiplication and addition operations are relatively low area or high-speed computations compared to division; this is the case in current FPGA technology. Secondly, iterative methods are highly suited to the determination of search directions for optimization problems, since early termination of an iterative method provides a mechanism to trade accuracy for computational throughput, as used to good effect in truncated-Newton methods [17].

A. Conjugate Gradient

The Conjugate Gradient (CG) algorithm is a Krylov subspace method that solves $Qy = r$ by repeatedly performing matrix-vector multiplications involving Q. Starting with an value of y_{k-1}, each iteration of the algorithm iteratively improves the solution to y_k by chosing y_k from the subspace spanned by $\{p_0, Qp_0, \ldots, Q^{k-1}p_0\}$ to minimize the Q-norm of the residual, given by $||p_k||_Q$, where $p_k = Qy_k - r$, $||x||_Q = x^T Qx$. The subspace is built iteratively, and requires little storage, due to the short recurrences used. Moreover, the main computation involved is repeated multiplication by the matrix Q, an operation known to map well to FPGA hardware [14]. The overall CG algorithm is summarized in Fig. 2.

Input: Matrix Q, Vector r, Tolerance ϵ
Output: y such that $||Qy - r||_2 \leq \epsilon||b||_2$

$d \leftarrow b$ (cg1)
$p \leftarrow b$ (cg2)
$\delta_0 \leftarrow p^T p$ (cg3)
$\delta_{new} \leftarrow \delta_0$ (cg4)
do
  $q \leftarrow Qd$ (cg5)
  $\alpha \leftarrow \delta_{new}/d^T q$ (cg6)
  $y \leftarrow y + \alpha d$ (cg7)
  $p \leftarrow p - \alpha q$ (cg8)
  $\delta_{old} \leftarrow \delta_{new}$ (cg9)
  $\delta_{new} \leftarrow p^T p$ (cg10)
  $\beta \leftarrow \delta_{new}/\delta_{old}$ (cg11)
  $d \leftarrow r + \beta d$ (cg12)
while $\delta_{new} > \epsilon^2 \delta_0$ (cg13)

Fig. 2. The Conjugate Gradient Algorithm

B. Hardware Design

The most computationally intensive operation within CG is the matrix-by-vector multiplication in (cg5). To obtain scalable performance, we may implement this computation by sequentially operating on each matrix row in turn, however each constituent inner product is fully unrolled and parallelised; Fig. 3 presents a schematic of a portion of the circuit implemented. The parallelism achieved is two-fold: in addition to the use of physically distinct computational units for each operation in an inner product, the inner product computation is itself deeply pipelined. As a result, a new pair of vectors can be introduced being introduced at each clock cycle. As a result, this implementation completes a CG iteration every n cycles (to within an additive constant [14]).

However, the latency of one CG iteration is given by

$$\text{latency}(n) = 7n + 36\lceil \log_2 n \rceil + 127 \qquad (2)$$

where the linear growth comes from the row-by-row sequential processing, the logarithmic growth comes from the addition tree in the inner-product computation, and the constants are due to the pipeline depths of the components. The discrepancy between a throughput of one iteration every n cycles and the latency given in (2) is used to our advantage, by using the slack to operate on multiple different matrix/vector pairs in a round-robin pipelined fashion. The total number of linear systems that can be processed simultaneously by the pipeline is given by the ratio of (2) to the

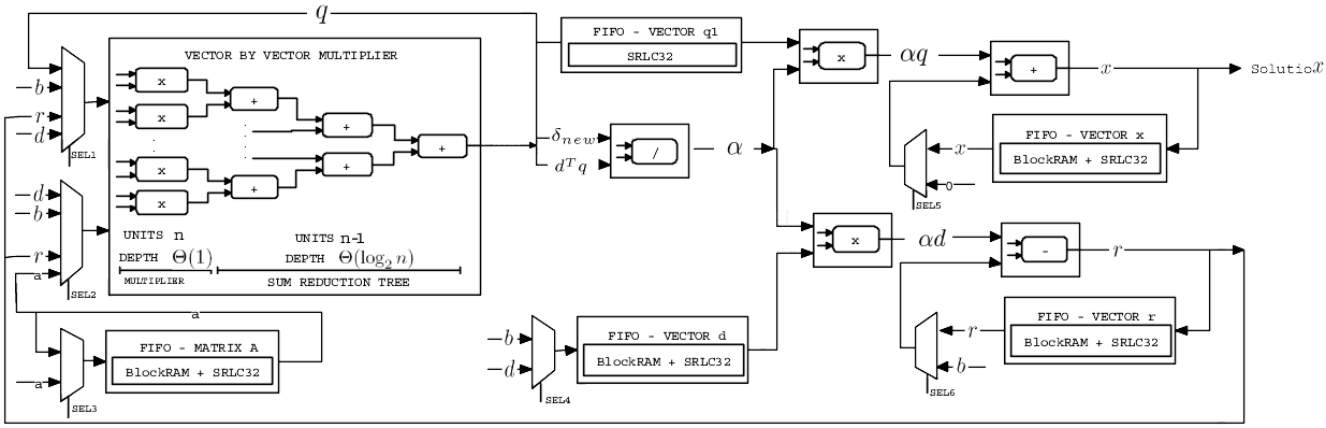

Fig. 3. Outline circuit schematic for a FPGA-based CG implementation [14].

$O(n)$ throughput, which is overall a $O(1)$ function that, for this implementation, converges to 8 for large n.

One of the major advantages of this scheme is its scalable FPGA input/output (I/O) bandwidth. The conjugate gradient algorithm completes in at most n iterations under infinite precision, and in $\Omega(n)$ iterations under finite precision [15][1]. As a result, the data transfer bandwidth required is a $O(1)$ function, *i.e.* approaches a constant for large n and is, in practice, well within existing FPGA I/O limitations.

C. Results

The proposed FPGA hardware design for the CG algorithm has been implemented, and the overall computational throughput given by the combination of parallelization and pipelining is illustrated in Fig. 4, which compares the processing time for each CG iteration, on the FPGA and a CPU. The three CPU curves are for an AMD Opteron 1220 - one shows the peak theoretical performance achievable, another shows the measured performance when running the highly-optimized ATLAS basic linear algebra subprogram library [18], and the third shows the performance of a naive direct C implementation of the algorithm.

The FPGA is the Virtex 5 330 from Xilinx [19]. Three curves are shown for the FPGA implementation. The first one represents the case where the computational pipeline contains only a single problem - in this case there is not enough data to keep the FPGA computational units active at all clock cycles. The intermediate curve shows the pipeline with 8 problems, and the third curve illustrates the performance of the FPGA when filled with however many problems are required to keep the units active - in this case the number of problems required varies with matrix order, and is shown underneath the curve. Unlike the microprocessor implementation, where measured and peak performance differ, mainly due to cache miss effects, the FPGA has completely predictable timing.

[1]In this paper, we use the following asymptotic notation common to the computational complexity literature. A function $f(n)$ is $O(g(n))$ iff $\exists c \in \mathbb{R}^+ \; \exists n_0 \in \mathbb{Z} \; \forall n > n_0 (f(x) \leq cg(n))$. A function $f(n)$ is $\Omega(g(n))$ iff $\exists c \in \mathbb{R}^+ \; \exists n_0 \in \mathbb{Z} \; \forall n > n_0 (f(x) \geq cg(n))$.

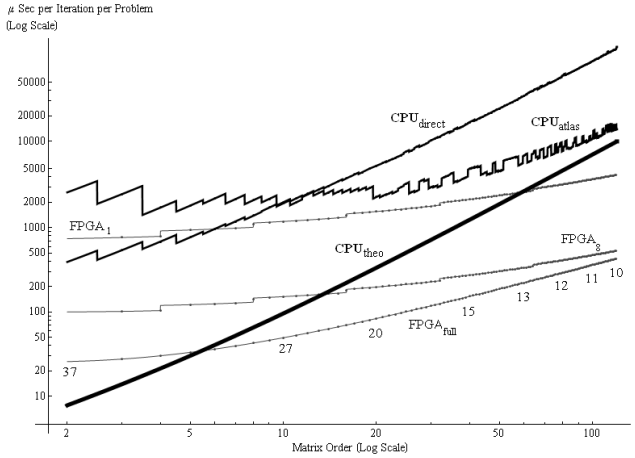

Fig. 4. Time required for a CG iteration as a function of matrix order on a CPU and FPGA.

It is clear from these results that the FPGA performance quickly overtakes the microprocessor, once enough parallelism is available to extract from the problem. Indeed, by the time one is operating with matrices of order 70 or greater, even if there is only one problem available for the FPGA, it outperforms even the peak theoretical capability of the CPU.

V. NON-STANDARD NUMBER REPRESENTATIONS

The results from the previous section use the IEEE standard single-precision floating point (approximate) representation of real numbers. In this representation, a real number x is represented as $x = (-1)^s \cdot 1.m \cdot 2^{e-127}$, where $s \in \{0, 1\}$ is the sign bit, m is the mantissa, represented as a 23-bit value, and e is the exponent represented as an 8-bit unsigned integer. The IEEE standards of single and double precision floating point are by far the most commonly implemented real number representation. Such standards are used in the development of general purpose computer architectures for two main reasons: repeatability and ease of use. For desktop computing applications, it may be important when migrating

software from one architecture to another, that the results achieved are *identical*, *i.e.* every bit of every floating-point number resulting from a computation is the same. In an embedded application, such as control, this is not generally a concern. We are aware that inaccuracy exists in the result of any numerical computation, primarily due to roundoff error, and our goal should be to quantify this inaccuracy and keep it under whatever limits are acceptable for the target application.

While it is entirely possible to implement IEEE standard arithmetic within FPGA hardware and obtain high performance results, as the previous section has illustrated, the above perspective raises the question of whether IEEE standard arithmetic is the most efficient choice for a customised architecture. It is clear that, since we have customised all other aspects of the computer architecture, there is no reason that we should not aim for a number representation that is also customised to the model predictive control application.

In a general purpose microprocessor, if an IEEE double precision unit is available, it makes little sense to use a precision less than this. However, the hardware required to implement a computation using k bits of mantissa scales asymptotically quadratically with k, and in practice between linearly and quadratically in k, depending on the underlying architecture. As a result, a reduction in k can impact significantly on the number of different parallel computational units that can fit within a fixed silicon area, and hence on the performance of an algorithm.

We may exploit this opportunity one step further, and take advantage of the spatial parallelism offered by the FPGA architecture: there is no reason why different computational units cannot operate with different precisions. The problem may then be cast as an optimization: maximize the performance of the circuit, as a function of the different mantissa widths, subject to acceptable numerical behaviour. The objective function, the performance of the circuit, can be modelled relatively straight-forwardly, once a hardware architecture, such as that in Fig. 3 has been developed. The constraints are far harder to evaluate: typical metrics in signal processing, such as signal-to-quantization (SQNR) noise ratio [20] are not particularly meaningful in the context of model predictive control, nor are they simple to calculate, as even SQNR is a highly nonlinear and data-dependent function of the individual roundoff errors within a contrained optimization algorithm. Within the limited context of linear, time-invariant DSP algorithms, this *multiple word-length* approach has been shown to provide speed increases of a factor of two over the *optimal* uniform word-length across the circuit [20], and many-fold over an IEEE standard.

VI. CHALLENGES FOR MPC

We have seen that the performance achievable by FPGA-based MPC is likely to significantly out-strip traditional microprocessor-based solutions, by an order of magnitude. However, much greater potential exists by making use of the flexibility of the FPGA hardware for implementing non-standard number representations and customising the MPC algorithms to the strengths of the hardware. This potential currently remains untapped, although there have been some point solutions [21].

I would pose a number of challenges in order to fully realise the potential of FPGA-based computation for MPC applications:

A. Accuracy, Parallelism, and Operation Count

There are, in general, many potential algorithms for solving a numerical problem. Often they differ in operation count and the accuracy and stability of their results with respect to small perturbations (such as those caused by roundoff error). A classical example is the matrix multiplication scheme proposed by Strassen [22], which computes the multiplication of two $n \times n$ matrices in $O(n^{2.81})$ operations, unlike the direct inner-product matrix multiplication, which takes $\Omega(n^3)$ operations. It is known, however, that roundoff errors accumulate more in Strassen's scheme [23].

If the number representation under which the problem is solved is fixed *a priori*, the algorithm is to be executed on a sequential machine, and cache effects can be ignored, then the selection of an appropriate algorithm is relatively trivial. One must simply list the algorithms in descending order of computation count, and select the first one with a sufficient accuracy for the task.

In modern architectures, whether manycore or FPGA-based, the picture is far more complex: one must consider each algorithm in terms of the accuracy, the parallelism, and the operation count they provide *as a function of the number representation chosen*. Strassen's algorithm may, for example, be preferable even for high-accuracy applications, since more accuracy can be given to the computations by using a non-standard number representation, so long as the parallelism that can be extracted is equal to that of the classical algorithm.

We are some way from making such decisions in an automated manner, and significant progress must be made here before we are to reap the full benefits of modern architectures.

B. Closed-loop impact of roundoff error

From an MPC perspective, the impact of roundoff error within the computation of the optimal control sequence must be studied within the context of its impact on the closed-loop system. An acceptable roundoff error is one that does not cause instability and does not impact significantly on the optimality of the control sequence. From this perspective, the impact of roundoff error could be considered to be similar to that of other forms of error, such as early termination of interior point methods, studied in [24]. Once an appropriate metric for the impact of roundoff error has been determined, we may revisit the optimization problem of determining a suitable numerical precision in order to maximize system performance.

C. Filling the pipeline

The architecture presented in Section IV as an example is deeply pipelined, and only achieves full efficiency if there

are at least eight distinct systems of linear equations to be solved in parallel. From the algorithm designer's perspective, we may view this as an architecture that takes essentially the same time to solve one system of linear equations as to solve eight. An important question is how to expose this extra computational power to the development of algorithms for MPC. Might it be possible, for example, to leverage recent work in multiplexed MPC [25] in order to achieve this goal by breaking down a large MPC problem into a series of smaller problems?

VII. CONCLUSIONS

This paper has aimed to provide a tutorial introduction into the new developments in computer architecture, with a particular focus on customised computer architectures built around field-programmable gate arrays. Such architectures hold out much promise for implementing sophisticated numerical algorithms within a hard real-time environment. As a result, model predictive control applications are likely to benefit. The impact of a speed improvement in MPC algorithms will be the potential to apply MPC in areas with fast dynamics where, until now, the computational load has appeared too great [26].

Common to these emerging parallel architectures is the requirement to reason about accuracy, whether due to the performance gap between single and double precision floating point exhibited by manycore architectures, or due to the aggressive silicon area dependence on precision exhibited in FPGA and ASIC-based computation.

Also common to these architectures is the need to extract large degrees of parallelism in order to keep deeply pipelined computational units running ar high efficiency, motivating potential reformulations of MPC problems in ways that can best expose this parallelism to the architecture.

In order to make the most of the promise of these architectures for control applications, control theorists will need to work with digital electronics designers and computer architects, to find appropriate realisations of control algorithms, spawning a new and exciting area of inter-disciplinary research.

VIII. ACKNOWLEDGMENTS

The author gratefully acknowledges the work of Mr. Antonio Roldao Lopes and Dr. Alastair Smith in preparing some of the figures in this paper, and also acknowledges the fruitful discussions discussions on this subject with Dr. Eric Kerrigan, Prof. Ling Keck-Voon, Prof. Jan Maciejowski, Prof. David Mayne, Mr. David Boland, Mr. Antonio Roldao Lopes, and Mr. Amir Shahzad.

REFERENCES

[1] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams, and K. Yelick, "The landscape of parallel computing research: A view from berkeley," UC Berkeley, Tech. Rep. UCB/EECS-2006-183, 2006.

[2] nVidia, "GeForce GTX 280," http://www.nvidia.com/object/geforce_gtx_280.html.

[3] A. Buttari, J. Dongarra, J. Langou, J. Langou, P. Luszczek, and J. Kurzak, "Mixed precision iterative refinement techniques for the solution of dense linear systems," *International Journal of High Performance Computing Applications*, vol. 21, no. 4, pp. 457–466, November 2007.

[4] N. Metropolis, J. Howlett, and G. Rota, Eds., *A History of Computing in the Twentieth Century*. Academic Press, 1980.

[5] H. Kung, "Why systolic architectures?" *IEEE Computer*, vol. 15, no. 1, pp. 37–46, 1982.

[6] O. Storaassli and D. Strenski, "Accelerating genome sequencing 100x with FPGAs," in *Proc. Reconfigurable Systems Summer Institute*, 2007.

[7] D. Boland and G. A. Constantinides, "An FPGA based implementation of the MINRES algorithm," in *IEEE International Conference on Field-Programmable Logic and Applications*, 2008, pp. 379–384.

[8] I. Pournara, C. Bouganis, and G. A. Constantinides, "FPGA-accelerated reconstruction of gene regulatory networks," in *IEEE International Conference on Field-Programmable Logic and Applications*, 2005, pp. 323–328.

[9] J. M. Maciejowski, *Predictive Control with Constraints*. Prentice Hall, 2002.

[10] T. A. Johansen, W. Jackson, R. Schreiber, and P. Tøndel, "Hardware synthesis of explicit model predictive controllers," *IEEE Transactions on Control Systems Technology*, vol. 15, no. 1, pp. 191–197, January 2007.

[11] S. J. Wright, *Primal-Dual Interior Point Methods*. Society for Industrial and Applied Mathematics, 1997.

[12] C. Rao, S. J. Wright, and J. Rawlings, "Application of interior point methods to model predictive control," *Journal of Optimization Theory and Applications*, vol. 99, no. 3, pp. 723–757, 1998.

[13] L. Zhuo and V. Prasanna, "High performance linear algebra operations on reconfigurable systems," in *Proc. Supercomputing*, 2005.

[14] A. Roldao and G. A. Constantinides, "A high throughput fpga-based floating point conjugate gradient implementation," in *Proc. Applied Reconfigurable Computing*, 2008, pp. 75–86.

[15] G. Meurant, *The Lanczos and Conjugate Gradient Algorithms from theory to Finite Precision Computation*. Society for Industrial and Applied Mathematics, 2006.

[16] K. Turkington, K. Masselos, G. A. Constantinides, and P. Leong, "FPGA based acceleration of the LINPACK benchmark: A high level code transformation approach," in *Proc. IEEE International Conference on Field-Programmable Logic and Applications*, 2006, pp. 1–6.

[17] S. Nash, "A survey of truncated-newton methods," *Journal of Computational and Applied Mathematics*, vol. 124, pp. 45–59, 2000.

[18] "Automatically tuned linear algebra software," http://math-atlas.sourceforge.net/.

[19] "Virtex5 family overview - LX , LXT, and SXT platforms," http://direct.xilinx.com/bvdocs/publications/ds100.pdf.

[20] G. A. Constantinides, P. Cheung, and W. Luk, "Wordlength optimization for linear digital signal processing," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 22, no. 10, October 2003.

[21] L. Bleris, J. Garcia, M. Kothare, and M. Arnold, "Towards embedded model predictive control for system-on-a-chip applications," *Journal of Process Control*, vol. 16, pp. 255–264, 2006.

[22] V. Strassen, "Gaussian elimination is not optimal," *Numer. Math.*, vol. 13, pp. 354–356, 1969.

[23] B. Dumitrescu, "Improving and estimating the accuracy of strassen's algorithm," *Numer. Math.*, vol. 79, pp. 485–499, 1998.

[24] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," in *Proc. IFAC World Congress*, 2008.

[25] K.-V. Ling and J. Maciejowski, "Multiplexed model predictive control," in *Proc. IFAC World Congress*, 2005.

[26] T. Keviczky and G. Balas, "Receding horizon control of an f-16 aircraft: A comparative study," *Control Engineering Practice*, vol. 14, no. 9, pp. 1023–1033, September 2006.