Implementation of a particle filter on a GPU for nonlinear estimation in a manufacturing remelting process

Felipe Lopez[1], Lixun Zhang[2], Joseph Beaman[1] and Aloysius Mok[2]

Abstract— **This paper discusses the use of modern methods for estimation in Vacuum Arc Remelting, a manufacturing process used in the production of specialty metals for aerospace applications. Accurate estimation in this process is challenging because the system is nonlinear and all available measurements are corrupted with noise. Particle filters are nonlinear estimators that sample a set of points, called particles, in the state space to construct discrete approximations of probability density functions. Real-time issues arise when using these methods in systems with low signal-to-noise ratios because of the large number of particles required to reach acceptable accuracy. In these cases, the throughput of the particle filter becomes critical, and parallelization becomes a necessity. This paper presents the implementation of a particle filter using a GPU with NVIDIA's CUDA technology, whose large number of processor cores allows massive parallelization.**

I. INTRODUCTION

Detection of manufacturing anomalies can be a complex task and requires accurate knowledge of the process. Traditionally, models used for process monitoring and control have been simplified representations of the physics of the process, while high-dimensional nonlinear models have often been used offline due to their computational cost. Nonetheless, there are manufacturing applications whose physical phenomena are not accurately described by low-order linear models and high-dimensional nonlinear ones are required.

In process monitoring, measurements obtained from the process are used along with physics-based models. Information from both sources is fed to an observer that estimates the state of the system. In order to preserve the accuracy of the model, simplifications such as linearity or normality are not admissible. Sequential Monte Carlo (SMC) methods, nonlinear algorithms based on Bayesian updates, are used for estimation. The particle filter, a well-known SMC method, has been used for offline state and parameter estimation in a broad spectrum of applications [1].

In online applications, the time it takes to perform a task is critical for the correct functioning of the system. It is crucial that estimation is performed faster than the time step determined for the process, and as a result, computations often have to be accelerated. The study of computer architectures, especially parallel architectures, used to accelerate algorithms for real-time operation is an active area of research. Particular attention is given to process control applications; in estimation [2], [3]; and control [4]. A study of computer architectures for acceleration of particle filtering in remelting processes was performed using field-programmable gate arrays (FPGAs) in a low-order nonlinear thermal model [5], but to the best of the knowledge of the authors there had not been any other applications to manufacturing processes.

This paper presents a manufacturing process where defect formation is dependent on solidification, which is described by a relatively high-dimensional nonlinear system. Particle filtering is too slow for process control when implemented on a personal computer. However, the same algorithm is accelerated when implemented on a graphic processing unit (GPU), meeting the time constraints for real-time control. The paper is organized as follows: Section II presents an overview of the process to be monitored. Section III describes the particle filtering algorithm. Section IV presents the implementation of the particle filter on a CPU and on a GPU architecture. Section V discusses the results, and section VI concludes the paper.

II. OVERVIEW

A. Vacuum Arc Remelting

Remelting processes for specialty metals, used to improve microstructural quality, are a potential area of interest for more accurate technologies in defect detection. Vacuum Arc Remelting (VAR) is usually the last step in the production of such ingots. It is extremely important to prevent segregation defects in VAR because they cannot be removed by posterior heat treatment. In this process, a continuous arc is struck between an electrode and a solidifying ingot, which results in metal melting off the electrode and falling to a liquid pool atop the new ingot, where it solidifies. By performing this procedure in a vacuum, oxides and other volatile compounds are removed from the material improving its homogeneity.

Control of the solidification front is expected to result in improved microstructures [6]. However, such a task requires state estimates that accurately track the thermal dynamics of the solidification front.

B. Dynamic model

A dynamic model of the process must include two subsystems: the melting of the electrode and the solidification of the ingot. The two subsystems are coupled by the fraction of supplied power that enters the liquid pool, called the melting efficiency. The electrode subsystem is nonlinear and

*This work was supported in part by NSF Grant No. 1239343, via the CPS initiative.

[1]Felipe Lopez and Joseph Beaman are with the Department of Mechanical Engineering, University of Texas at Austin, Austin, TX 78705, USA felipelopez@utexas.edu, jbeaman@mail.utexas.edu

[2]Lixun Zhang and Aloysius Mok are with the Department of Computer Science, University of Texas at Austin, Austin, TX 78705, USA lxzhang@cs.utexas.edu, mok@cs.utexas.edu

includes measurements with a low signal-to-noise ratio, such as electrode gap. Furthermore, the dynamics of the process are highly dependent on the melting current and the way it is distributed in crucible. Unfortunately, due to the way in which electrical energy is transferred in the system, current measurements are noisy. Estimation is not any easier for the ingot dynamics, mainly because it is not possible to measure the solidification front during the melt. A finite volume model running in parallel to the process has been used for such a task [7].

In this model, the state vector $\mathbf{x}_k$ is given by the thermal boundary layer in the melting electrode, electrode gap, ram position, electrode mass, melting efficiency, liquid pool depth at the centerline and mid-radius, melting current and helium pressure. There are only two variables in the inputs vector $\mathbf{u}_k$: commanded current and ram velocity. The output vector $\mathbf{y}_k$ is given by the electrode gap, ram position, current, electrode mass, centerline and mid-radius pool depths, and helium pressure.

The state-space model, denoted by the nonlinear functions f and g, is stochastic due to the inherent uncertainty in the model, model parameters, and measurements. Noise terms are arranged in the form of process noise ($\mathbf{v}_k$), and measurement noise ($\mathbf{w}_k$).

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}, \mathbf{v}_{k-1}) \tag{1}$$

$$\mathbf{y}_k = g(\mathbf{x}_k, \mathbf{w}_k) \tag{2}$$

The model can also be given in the form of conditional probabilities $\mathbf{x}_k|\mathbf{x}_{k-1} \sim f(\mathbf{x}_{k-1}, \mathbf{u}_{k-1})$ and $\mathbf{y}_k|\mathbf{x}_k \sim g(\mathbf{x}_k)$ where f and g are appropriate probability measures.

III. PARTICLE FILTER

In Bayesian estimation the goal is to build the posterior probability distribution function of the state vector at each time given all the measurements available. In other words, the goal is to construct $p(\mathbf{x}_k|\mathbf{y}_{1:k})$.

If the probability density $p(\mathbf{x}_{k-1}|\mathbf{y}_{1:k-1})$ is available, one can follow a two-step approach to construct the conditional density for the following time step. First, at the prediction stage, the prior is calculated by

$$p(\mathbf{x}_k|\mathbf{y}_{1:k-1}) = \int f(\mathbf{x}_k|\mathbf{x}_{k-1}) p(\mathbf{x}_{k-1}|\mathbf{y}_{1:k-1}) d\mathbf{x}_{k-1}$$

and at the update stage, the posterior is given by

$$p(\mathbf{x}_k|\mathbf{y}_{1:k}) = \frac{g(\mathbf{y}_k|\mathbf{x}_k) p(\mathbf{x}_k|\mathbf{y}_{1:k-1})}{p(\mathbf{y}_k|\mathbf{y}_{1:k-1})}$$

where $p(\mathbf{y}_k|\mathbf{y}_{1:k-1}) = \int g(\mathbf{y}_k|\mathbf{x}_k) p(\mathbf{x}_k|\mathbf{y}_{1:k-1}) d\mathbf{x}_k$.

A particle filter recursively approximates the random variable $\mathbf{x}_k|\mathbf{y}_{1:k}$ by particles $\{\mathbf{x}_k^i\}_{i=1}^N$ with discrete probability masses $\{\mathbf{w}_k^i\}_{i=1}^N$. The set of samples, $\mathcal{S}_k = \left\{\mathbf{x}_k^i, \mathbf{w}_k^i\right\}_{i=1}^N$, is said to be properly weighted when summation over the particles approximates expectation under the posterior $\mathcal{P}_k$; i.e.,

$$\mathrm{E}_{\mathcal{S}_k}\left[h(\mathbf{x}_k)\right] \equiv \sum_{i=1}^{N} \mathbf{w}_k^i h(\mathbf{x}_k^i) \xrightarrow{N\to\infty} \mathrm{E}_{\mathcal{P}_k}\left[h(\mathbf{x}_k)\right] \tag{3}$$

for a sufficiently smooth function h.

Particle filters treat the discrete distribution generated by the particles as the true filtering density, and use it to construct an approximation of the prediction density $\mathbf{x}_k|\mathbf{y}_{1:k-1}$ using the prior f.

$$\hat{p}(\mathbf{x}_k|\mathbf{y}_{1:k-1}) = \sum_{i=1}^{N} f(\mathbf{x}_k|\mathbf{x}_{k-1}^i) \mathbf{w}_{k-1}^i \tag{4}$$

The distribution can be combined with the likelihood to result in the posterior distribution, up to a proportionality.

$$\begin{aligned}\hat{p}(\mathbf{x}_k|\mathbf{y}_{1:k}) &\propto g(\mathbf{y}_k|\mathbf{x}_k) \sum_{i=1}^{N} f(\mathbf{x}_k|\mathbf{x}_{k-1}^i) \mathbf{w}_{k-1}^i \\ &\propto \sum_{i=1}^{N} f(\mathbf{x}_k|\mathbf{x}_{k-1}^i) [g(\mathbf{y}_k|\mathbf{x}_k) \mathbf{w}_{k-1}^i]\end{aligned} \tag{5}$$

The density defined in equation (5) is approximated by propagating particles $\{\mathbf{x}_{k-1}^i\}_{i=1}^N \to \{\mathbf{x}_k^i\}_{i=1}^N$ following the prior, and then using the likelihood to update their weights following $\mathbf{w}_k^i = \mathbf{w}_{k-1}^i g(\mathbf{y}_k|\mathbf{x}_k^i) / \sum_{i=1}^N \mathbf{w}_{k-1}^i g(\mathbf{y}_k|\mathbf{x}_k^i)$. It should be noted that the support of the distribution is completely defined by particles obtained from the approximation of $p(\mathbf{x}_k|\mathbf{y}_{1:k-1})$, and the likelihood is used only to adjust the weights.

Increasing variability as time progresses impairs the performance of the particle filter in a problem commonly referred to as degeneracy. To avoid degeneracy, resampling is performed when the effective sample size (ESS)[1] falls below a threshold $N_T = N/2$: higher-weight particles usually are duplicated, with the duplicity determined by the weight of the particle and the particular resampling scheme, while lower-weight particles are eliminated. There are several resampling methods, among which systematic resampling has been reported to have the highest quality [8]. The particle filter, also known as the *Sampling Importance Resampling* (SIR) algorithm, is shown in Algorithm 1.

IV. IMPLEMENTATION OF THE PARTICLE FILTER

Particle filters have been successfully applied in real-time control applications [9], [10]. Although promising, the particle filter algorithm has some weaknesses as a result of the discrete approximations of continuous probability distributions. The variance of particle filter estimates is approximately proportional to $\mathrm{E}\left[\mathbf{w}_k^2\right]/N$, and therefore, high variance in the estimates should be expected when the particle weights $\{\mathbf{w}_k^i\}_{i=1}^N$ are variable. Such a behavior appears when the likelihood function is highly peaked compared to the prior [11]. Also, it is possible that some regions of the state space are not explored with enough particles, while some others have them closer together than necessary. Ref. [12] shows that the minimum number of particles required to bound the variance of the posterior space is exponential in state dimension.

[1]The ESS can be seen as the number of particles, independently sampled from the true posterior distribution, that would result in estimates of equivalent accuracy.

Algorithm 1 SIR Particle Filter.
$[\{\mathbf{x}_k^i, \mathbf{w}_k^i\}_{i=1}^N] = \text{SIR}([\mathbf{y}_k, \{\mathbf{x}_{k-1}^i, \mathbf{w}_{k-1}^i\}_{i=1}^N])$

for $i = 1$ to N **do**
  Sample $\mathbf{x}_k^i \sim f(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}, \mathbf{v}_{k-1})$
end for
for $i = 1$ to N **do**
  Compute $\mathbf{w}_k^i = g(\mathbf{y}_k|\mathbf{x}_k^i)\mathbf{w}_{k-1}^i$
end for
Compute total weight: $t_k = \sum_{i=1}^N \mathbf{w}_k^i$
for $i = 1$ to N **do**
  Normalize weight: $\mathbf{w}_k^i = \mathbf{w}_k^i / t_k$
end for
Compute effective sample size: ESS $= (\sum_{i=1}^N \mathbf{w}_k^{i\,2})^{-1}$
if ESS $< N/2$ **then**
  Initialize cumulative weight: $c_1 = \mathbf{w}_k^1$
  for $i = 2$ to N **do**
    Compute cumulative weights: $c_i = c_{i-1} + \mathbf{w}_k^i$
  end for
  Generate a random number: $u \sim U[0, 1)$
  $j = 1$
  for $i = 1$ to N **do**
    $u_i = \frac{(i-1)+u}{N}$
    while $c_j < u_i$ **do**
      $j = j + 1$
    end while
    Duplicate: $\mathbf{x}_k^i = \mathbf{x}_k^j$
  end for
  for $i = 1$ to N **do**
    Uniformize weight: $\mathbf{w}_i = N^{-1}$
  end for
end if

Due to these potential problems, the performance of the particle filter as an estimator was tested offline before its implementation in a real-time control loop. These simulations were intended to determine the accuracy of the estimator, the minimum number of particles required, and the computation time. The particle filter was implemented using the SMCTC library, developed by Johansen in C++ [13]. The filter was tested with experimental data correspondent to a melt of Alloy 718 from an electrode of 17" into a 20" ingot, using $N = 2^{24}$ particles and noise strengths common in VAR practice. The experiment lasted for 18 hours but only 50 minutes are shown in Fig. 1, where the noisy measurements are compared to the maximum likelihood estimates obtained from the sampled particles.

Accurate tracking can be observed for ram position, electrode mass, and helium pressure; where accurate measurements are available from the furnace. Electrode thermal boundary layer and melting efficiency, in the other hand, are states that cannot be measured directly from the furnace and estimates were inferred from other measurements and from the stochastic model. Also, particular attention must be given to electrode gap because furnace measurements are extremely noisy. In the physical model, electrode gap changes due to electrode melting and the translation of the electrode, both of which are slow processes. Therefore, knowledge from the physics of the process indicates that electrode gap does not change as fast as their measurements do, but it has much slower dynamics, as it is observed in the estimate returned by the particle filter.

Measurements of the geometry of the solidification front are known to be uncertain due to numerical and grid-related artifacts, and are combined with a linearization of the solidification dynamics of the ingot, which is uncertain as well. Liquid pool depth estimates and measurements follow alike dynamics, but there is a small offset due to mismatch between the stochastic model and the finite volume one.

It was observed that a large number of particles - i.e., at least tens of thousands but preferably at least hundreds of thousands - was required for tracking the process variables, partly due to the dimensionality of the system but mainly because of the low ESS, which resulted in resampling at every single iteration.

Data filtering was performed offline using time steps of 6 seconds[2]. The simulation was run on a personal computer with an Intel®Core i7 CPU 860 at 2.80 GHz x 8 processor, using Ubuntu 12.10 as the operating system. The C++ code was compiled with the g++ compiler without any optimization options. Particle filtering with $N = 2^{20}$ particles took almost as long as the melt, which leaves no room for control calculations or communication delays that may occur in an actual melt. Time constraints are even tighther in the case of smaller ingots (i.e., 8" in diameter), where the time step used in process control is usually close to 2 seconds, less than the time required for the particle filter to return an estimate. As a result, it is clearly necessary to accelerate the particle filter in order to make it suitable for online operation.

A parallel computer architecture, such as a General-purpose Graphics Processing Unit (GPGPU), is expected to accelerate the operation of the particle filter. NVIDIA's CUDA architecture is used in the implementation presented in this paper. The CUDA architecture is built upon an array of Streaming Multiprocessors (SMs). GeForce GTX TITAN, the graphics card used in this study, has 14 SMs. Since each SM consists of 192 Scalar Processor (SP) cores [14], there is a total of 2688 cores. The large number of processor cores allows massive parallelization.

CUDA extends the C language and introduces a new kind of functions, called *kernel* function, which run internally in the GPUs. The kernel function is executed concurrently by a number of threads. These threads are organized in *blocks*: there are one or more blocks, and each block consists of the same number of threads. One block can run on only one SM at a time; and if the number of blocks is larger than the number of SMs, the distribution of work across different SMs and the switch of blocks are automatically performed by a hardware scheduler.

The CUDA architecture comes with a hierarchy of memory. *Global memory* resides in the device memory, usually with large size but latency of 400-600 clock cycles, because

[2]The value was taken from usual industrial practice in ingots of similar size.

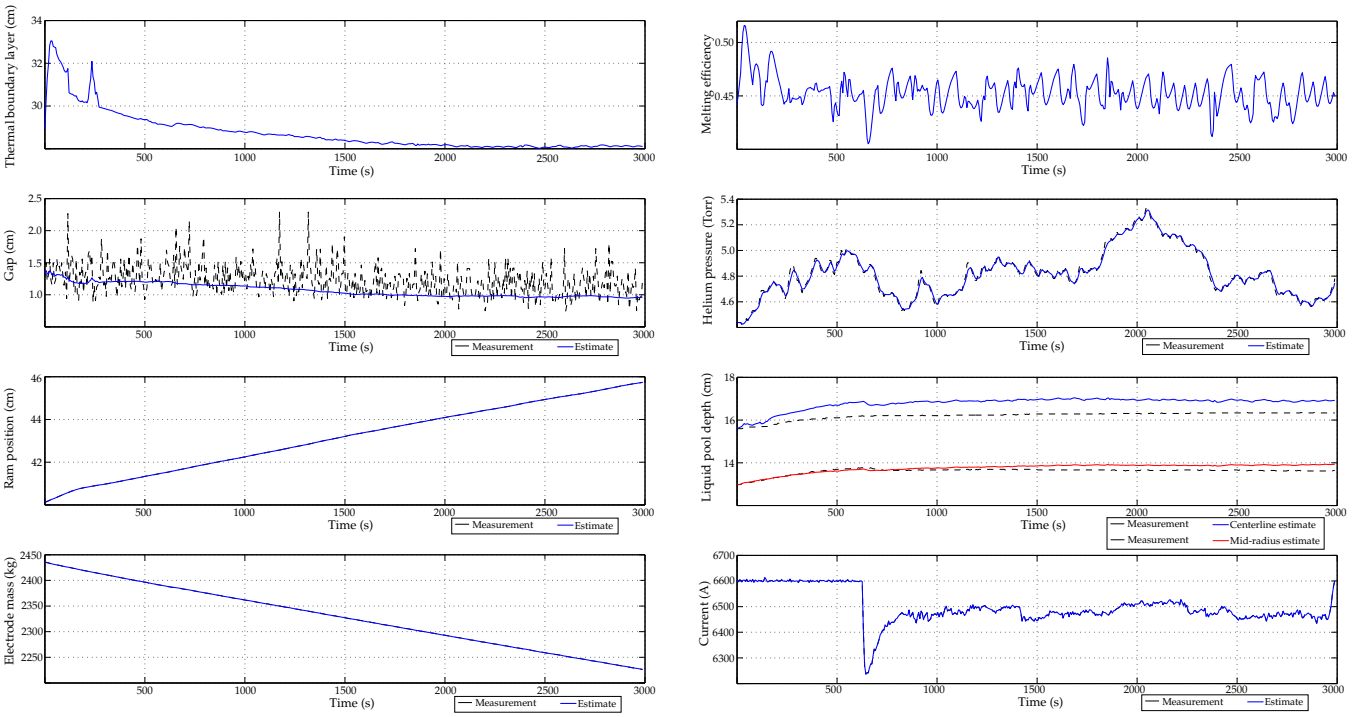


Fig. 1. Measurements and state estimates returned by the particle filter in VAR of Alloy 718 from an electrode of 17" into an ingot of 20".

it is off-chip and not cached. *Shared memory* resides on chip and has only one clock latency. It has a limited size; which can be configured as 32 KB, 48 KB or 64 KB for each SM. It has scope of all threads in a block, and lifetime of the block. CUDA also provides on-chip *registers* that have scope and lifetime of one thread, and off-chip *local memory* that has the same scope and lifetime as registers but is much slower. Whether automatic variables in a kernel function are stored in registers or local memory is decided by the CUDA compiler and determined by the number of registers.

Given this hierarchy, if a kernel function wants to write data that can be read later by another kernel function call, it must write in global memory. On the other hand, generally speaking, if a kernel function wants to do a local calculation that involves many accesses to the same data in global memory, it should first read the data to shared memory, do the calculation on shared memory and write the results back to global memory, if necessary.

The particle filter was implemented on the GPU using hardware and software listed in Table I. Random numbers were computed using the *CURAND library*, provided in Ref. [15], before each iteration. However, since the generation of random numbers on GPU could be done at the same time when the physical VAR model is functioning on the CPU, the latency of random number generation is hidden.

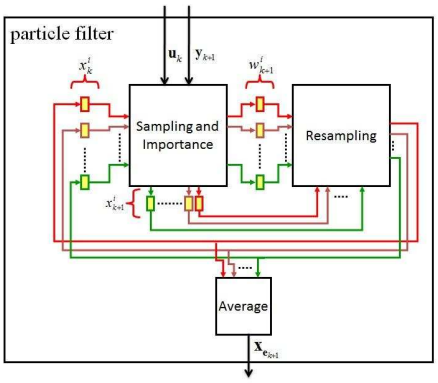
The particle filter was implemented using three modules (see Fig. 2(a)):

- **Sampling and Importance** In this module (see Fig. 2(b)), each particle i goes through the evolution equation, f, to propose a particle $\mathbf{x}_{k+1}^i$, and then its weight $\mathbf{w}_{k+1}^i$ is calculated at the *Likelihood and Weight* sub-module. Finally, the weights are normalized and output to the next module.
- **Resampling** This module resamples the particles according to their weights. Small-weight particles are replaced by the large-weight ones. To compute the cumulative weights, the parallel *scan then fan* scheme, provided in Ref. [16], is applied. Since the cumulative weights are monotonically increasing, this search is ideal for *binary search* algorithm, with $\mathcal{O}(logN)$ operations for each particle. It is important to notice that the **Resampling** module is executed only when the ESS falls below N_T, as suggested in the literature [1].
- **Average** This module calculates the maximum likelihood $\mathbf{x}_e$ to be used by the process controller, which is not included in this paper. The parallel reduction scheme, provided in CUDA SDK v5.5, is applied for such a task.

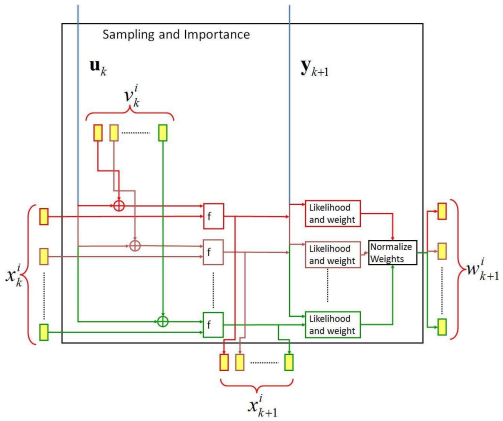
TABLE I
HARDWARE AND SOFTWARE USED FOR EVALUATION

GPU (CUDA)	
Model:	NVIDIA GeForce GTX TITAN
Driver Version:	320.57
Clock Speed:	928 MHz
BUS:	PCI Express 3.0
CUDA Driver/Runtime Vers.:	6.0/5.5
CUDA Computing Ability:	3.5
OS:	Windows 8 64-bit
Compiler:	Visual Studio 2012

Lines 1-6 in Algorithm 1 are parallel by nature, and can be carried out using one thread per particle. In this scheme,



(a) Particle filter



(b) Sampling and importance module of the particle filter

Fig. 2. Schematic of the GPU implementation of the particle filter.

the blocks are organized in one dimension. Suppose there are `num_block` blocks, each block contains `dim_block` threads, and for maximum parallelization each thread computes for one particle. The total number of particles $N =$ `num_block` $*$ `dim_block`. Then i-th thread in j-th block computes for particle $i + (j-1) *$ `dim_block`. Fig. 3 illustrates an example for f function, where $N = 512$, `num_block` $= 16$, and `dim_block` $= 32$. The numbers in global memory indicate the particle indexes, while the numbers in thread blocks indicate the thread indexes in each block.

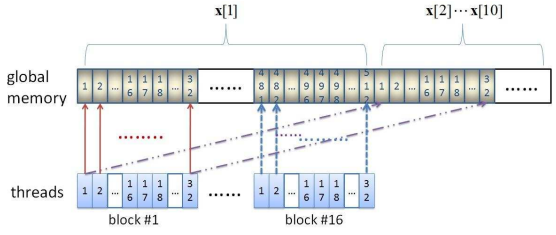


Fig. 3. Global memory access by one thread per particle. Same type of arrows indicate coalesced memory accesses that result in only *one* transaction.

Notice that the memory layout and access pattern in Fig. 3 are chosen for fast access to global memory. In CUDA, global memory access of 32 threads is coalesced into one single transaction if all the accessed words lie into the same data segment, otherwise n transactions are issued for n data segments[3][14]. In the example, all the accesses of 32 threads are coalesced within 128-byte segments, and thus are finished in *one* transaction. They are represented by the same type of arrow in the figure.

V. RESULTS

The elapsed time for one iteration, both for the CPU and the GPU implementations, is shown in Fig. 4 for $N = 2^i$, $i = 14, 15, \ldots, 23$ [4]. It is observed that the GPU implementation is consistently faster, being in average 16.2 times faster than its CPU counterpart for the same number of particles. The constraint for real-time operation, the time step for VAR in large ingots, is also shown in the figure. Particle filtering lasts longer than the process itself for $N > 10^6$ particles in the CPU implementation, preventing their use. The GPU implementation does not reach the time constraint even when $N = 2^{23}$ particles are used. The constraint for the GPU implementation is given by the amount of global memory available in the GPU, which is 4096 MB ($4,294,967,295$ bytes). The implementation of a particle filter with more than $N = 2^{23}$ particles would require a larger global memory.

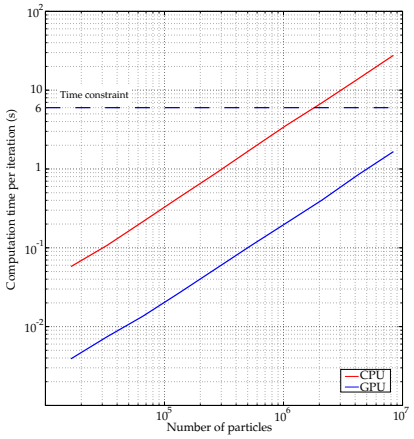


Fig. 4. Processing time for the particle filter implemented on a CPU and a GPU.

In the case of smaller VAR ingots, the time constraints are tighter, and change approximately proportionally to the square of the diameter. It is common to use time steps of 2 seconds in laboratory-scale melts, which would limit the number of particles to $N = 4 \times 10^5$ for a CPU implementation, while the GPU implementation is still unaffected by this constraint.

A low ESS was observed in the simulations. Such a phenomenon indicates that most of the particles proposed with the prior result in zero likelihood predictions given the observed measurements. The observed ratio between the ESS

[3]The size of data segment is 32 bytes if the accessed word size is 1-byte, 64 bytes for 2-byte word access, 128 bytes for 4-byte or 8-byte word access. In our program, since the accessed word size is 8-byte, the size of data segment is 128 bytes.

[4]The number of particles N was chosen to be a power of 2 for implementation in the GPU architecture.

and N was 0.04, much smaller than the resampling threshold of 0.5. Based on these observations one could argue that, although promising and fast, the SIR algorithm may not be the best choice for nonlinear estimation in VAR. Some other SMC algorithms that incorporate knowledge of the measure y_k when drawing proposals, such as the auxiliary particle filter [11], might be better suited. However, the structure shown in Fig. 2(b) would be different due to multinomial resampling being performed before drawing proposals.

VI. CONCLUSIONS

Optimal implementation of the particle filter for estimation in manufacturing processes requires a computer architecture that takes the parallel nature of the algorithm into account in order to accelerate its perfomance. A GPU with CUDA technology was used to meet real-time contraints for state estimation in a Vacuum Arc Remelting process. The GPU implementation was approximately 16 times faster than its CPU counterpart, proving to be fast enough for real-time operation both in VAR of large ingots and laboratory-scale ones. These results hold the promise of being extendable for similar manufacturing processes with faster dynamics and tighter time constraints, e.g. welding. The limitations of the proposed GPU program were analyzed as well. The number of particles cannot exceed $N = 2^{23}$ because of hardware constraints.

The accuracy of the particle filter is known to be superior to that of a Kalman filter and other estimators because it does not assume linearity nor normality. Unfortunately, particle methods are not as robust as Kalman-based techniques and some potential issues were presented in this paper, mainly due to the peaked nature of the likelihood compared to the prior. Also, compared to a Kalman filter, the particle filter returns the complete probability distribution of the state, which can be used by an stochastic predictive controller. An approach similar to the one given in this paper can be applied to other processes with low signal-to-noise ratios, or where linearization and order reduction result in a loss of accuracy in anomalies detection.

VII. ACKNOWLEDGMENT

The authors would like to thank David Evans and staff at Special Metals Corportation New Hartford for providing the data used in this paper. The experiment was intended to test the performance of a pool profile controller and does not represent usual practice of Special Metals Corporation.

REFERENCES

[1] A. Doucet and A. M. Johansen, *The Oxford Handbook of Nonlinear Filtering*. Oxford University Press, 2011, ch. A tutorial on particle filtering and smoothing: fifteen years later.

[2] L. Miao, J. Zhang, C. Chakrabarti, and A. Papandreou-Suppappola, "A new parallel implementation for particle filters and its application to adaptive waveform design," in *2010 IEEE Workshop on Signal Processing Systems (SIPS)*, October 2010, pp. 19–24.

[3] M. Bolic, P. M. Djuric, and S. Hong, "Resampling algorithms and architectures for distributed particle filters," *IEEE Transactions on Signal Processing*, vol. 53, no. 7, pp. 2442–2450, 2004.

[4] G. Constantinides, "Tutorial paper: Parallel architectures for model predictive control," in *Proceedings of the European Control Conference*, 2009, pp. 138–143.

[5] T. Lauzon, "Suitability of FPGA-based computing for cyber-physical systems," Master's thesis, The University of Texas at Austin, December 2009.

[6] T. J. Watt, E. M. Taleff, L. F. Lopez, J. Beaman, and R. Williamson, "Solidification mapping of a nickel alloy 718 laboratory VAR ingot," in *Proceedings of the 2013 International Symposium on Liquid Metal Processing & Casting*. The Minerals, Metals & Materials Society (TMS), 2013, pp. 261–270.

[7] J. Beaman, L. F. Lopez, and R. Williamson, "Modeling of the vacuum arc remelting process for estimation and control of the liquid pool profile," *ASME Journal of Dynamic Systems, Measurement, and Control*, vol. 136, no. 3, 2014.

[8] J. D. Hol, T. B. Schon, and F. Gustafsson, "On resampling algorithms for particle filters," in *2006 IEEE Nonlinear Statistical Signal Processing Workshop*, September 2006, pp. 79–82.

[9] R. Morales-Menéndez, N. de Freitas, O. De Freitas, and D. Poole, "Estimation and control of industrial processes with particle filters," in *American Control Conference*, 2003, pp. 579–584.

[10] D. Stahl and J. Hauth, "PF-MPC: Particle filter-model predictive control," *Systems & Control Letters*, vol. 60, no. 8, pp. 632–643, 2011.

[11] M. Pitt and N. Shepard, *Sequential Monte Carlo methods in practice*. Springer, 2001, ch. Auxiliary variable based particle filters, pp. 271–293.

[12] K. Choo and D. J. Fleet, "People tracking using hybrid Monte Carlo filtering," in *Proceedings of the Eighth IEEE International Conference on Computer Vision*, vol. 2, July 2001, pp. 321–328.

[13] A. M. Johansen, "SMCTC: Sequential Monte Carlo in C++," *Journal of Statistical Software*, vol. 30, no. 6, pp. 1–41, 4 2009. [Online]. Available: http://www.jstatsoft.org/v30/i06

[14] *CUDA C Programming Guide*, NVIDIA Corporation, 2013, version 5.5. [Online]. Available: http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf

[15] *CURAND Toolkit*, NVIDIA Corporation, version. [Online]. Available: http://docs.nvidia.com/cuda/curand/

[16] S. Sengupta, M. Harris, and M. Garland, "Efficient parallel scan algorithms for GPUs," NVIDIA Corporation, Tech. Rep. NVR-2008-003, December 2008.