A FPGA Accelerator for Real-Time 3D Non-Rigid Registration Using Tree Reweighted Message Passing and Dynamic Markov Random Field Generation

Michael Barrow[1], Steven M. Burns[2] and Ryan Kastner[1]
[1]University of California, San Diego [2]Intel Corporation, Hillsboro, OR USA

***Abstract*— Non-rigid 3D registration is a technique for matching 3D scans of a scene involving deformable objects. Augmented reality, gesture recognition, medical imaging, and many other computer vision and graphics applications require real-time registration to model deformable or articulated objects. Unfortunately, non-rigid registration is a computationally intensive problem that requires careful optimization to maximize throughput and latency. We present a FPGA+CPU accelerator for real-time non-rigid 3D registration based on Tree Reweighted Message Passing (TRW-S). We overcome memory bound issues and scheduling limitations of conventional TRW-S by dynamically generating the Markov Random Fields. This, along with a bevy of other architectural optimizations, allows us to almost saturate 1024 multipliers in a Arria 10 at 100MHz. We achieve a 600x speed up over baseline TRW-S and our registration architecture has up to 81x energy reduction over a software implementation of our algorithm. We demonstrate the performance of our system by performing real-time (20 scan per second) registration on a complicated surgical scene.**

***Index Terms*— Field Programmable Gate Array (FPGA), 3D Non-Rigid Registration, Sequential Tree Reweighted Message Passing (TRW-S), Dynamic Markov Random Field (MRF)**

I. INTRODUCTION

Image registration is the fundamental computer vision problem that matches two or more scans of an object to each other, i.e., given two scans X,$Y \subset \mathbb{R}^3$, registration seeks the mapping $f : Y \mapsto X$ that corresponds to the smallest transform T where: $X = T(Y)$. In other words, registration indicates the simplest way to manipulate Y so the scans share the same coordinate system.

Registration has many real time applications such as SLAM, Stereo reconstruction object tracking etc. [1], [2], [3]. A typical real time registration involves processing point cloud data from an input source (e.g., an RGBD camera) tens of times per second and mapping the 3D points from one frame to the next. We focus specifically on the 3D registration mapping function $f : Y \mapsto X$ in this work, which is a system bottleneck for real time performance in that it must perform a global optimization on the input data, and is a common step in all registration based applications.

Non-rigid 3D registration is a challenging but important class of registration where objects in a scan can deform non-rigidly. Fig 1 provides an example of a hand that deforms as the index finger flexes. Because the bones curl around each other, a non-rigid registration mapping function must be complex enough to consider all bones when registering an entire hand. As objects become less rigid, registration becomes more difficult as the number of articulation points increases.

(a) Hand scan X, fingers in open position

(b) Hand scan Y, index finger flexed

Fig. 1: Non-rigid registration of a flexing finger. Our mapping strategy is to use the invariant "geodesic distances" in both X and Y to correctly label Y's features such that they map to the same features in X

Non-rigid registration is computationally hard because a non-linear optimization procedure is required to find the correct pairing of the features in Fig 1. In contrast, rigid image registration does not need to find a unique pairing of features since a linear decomposition on any sub-set of features can reveal the one mapping that is valid for all points in a scan. This makes real time non-rigid registration much more challenging. It is extremely important to solve difficult non-rigid registration problems but unfortunately this has traditionally been hampered by the complexity of non-rigid mapping functions that can only achieve offline performance. For example, surgeons have long sought to use real time registration of soft tissue for image guidance applications [4], [5], [6], [7] and non-rigid registration has many uses in CV along a sliding scale of registration difficulty. Real time pose estimation is a simpler problem useful for many Human Computer Interaction applications such as virtual clothing, performance capture, autonomous driving and many others [8], [9], [10]. Therefore a general method for non-rigid registration with online performance is very desirable.

The contributions of this work are: 1) A *Real Time Non-rigid Registration Accelerator* which demonstrates performance of 20 scans per second on a difficult medical registration problem. A distinguishing feature of the accelerator is that it does not require "shape templates" or "object priors" and is therefore suitable for difficult and simple non-rigid registration problems without re-engineering. 2) The *"TRWL-S" Algorithm*. We describe several novel methods we used to transform a state of the art offline memory bound algorithm into an online compute bound one. 3) *Energy Efficient Architecture*. We analyze performance of our new algorithm on our architecture and find a speedup of 600X with an 81X reduction in system energy.

The paper is organized as follows: **Section** II summarizes the state of the art in online non-rigid registration. **Section** III details how our algorithm enables our accelerator to have higher performance than a state of the art baseline algorithm. **Section** IV describes the implementation of our novel registration architecture. **Section** V Evaluates the performance of the architecture. **Section** VI Provides our conclusions on this work.

II. Related Work

Non-rigid 3D registration is computationally intensive. To the best of our knowledge there is no system capable of solving the general problem in real time. Instead, systems achieve fast runtime with trade offs to simplify their problem.

For example, much work has focused on real time pose estimation where object priors simplify the registration problem. Object priors are a set of rules describing plausible configurations for a canonical object. Registration requires finding a transform of a scan that best fit these rules and only works on objects that adhere to these rules. This is a simpler problem than performing registation with unknown content. The most common priors are object skeletons which can be articulated into the poses of interest [11], [12], [13], [14]. These approaches cannot be applied to more complex registrations, e.g., they are inadequate for registering pliable objects that deform without points of articulation.

Another approach possible is to define a registration template from the data stream itself. Newcombe et al. [1] avoid pre-defined priors by taking one image scan to be a canonical frame. The work achieves real time performance by regularizing (linearizing) the fitting of input scans to the canonical frame. They demonstrate that more complex real time registrations are possible with this approach as compared with skeleton templates; however the linear optimizer overconstrains the shape matching problem making registration of very flexible objects difficult.

Dou et al. [9] propose another real time system that does not require templates and is more robust than Newcombe's to large deformations. The registration is done using a deformation graph [15]. Although the results are compelling, a major drawback is that the deformation graphs must be defined by the user, which limits the generality of the technique.

In contrast, global optimization registration methods are more general since they do not require any object priors. For example, Chen et al. [16] acheived accurate results by formulating the optimization problem as a Markov Random Field (MRF). Each entry in the tensor is a cost associated with pairing geodesic distances between features. While the quality is high, the optimizer run time is prohibitively slow for real time applications.

Real time global non-rigid registration is therefore possible if an optimizer exploring the MRF problem can do so in real time. In addition if the number of features "n" that can be described by the MRF is large enough, this approach would be suitable for difficult registration problems.

There have been recent advances in GPU, FPGA, and ASIC acceleration for efficient message passing based global optimizers which explore MRF's in linear time [17], [18], [2], [19]. Unfortunately these approaches are revealed to be memory bound as they must stream the MRF into the accelerated optimizer. Fig 2 illustrates accelerator memory bound for MRF problems sizes of Chen's approach[20]. Fig 2 shows it is not possible to solve MRF registration problems in real time for $n \geq 60$ with existing accelerators because the MRF cannot be loaded into the accelerator quickly enough.

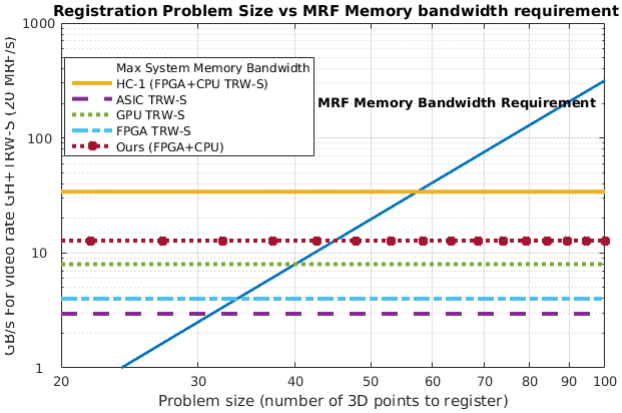


Fig. 2: MRF memory bandwidth scaling with problem size (n) [20]. No proposed global optimizer accelerator system (horizontal trends) is suitable for difficult non-rigid registration since the MRF must be streamed in. The memory bandwidth available for transferring a global registration problem MRF to the proposed accelerators [17], [18], [2], [19] is not sufficient for a moderately complex registration problem of n=60 feature points.

III. Registration Algorithm

Our Tree Reweighted Loaf Slice (TRWL-S) algorithm builds on a global non-rigid registration baseline described by Chen et al. [16] which registers scans using geodesic distances (see Fig 1). Fig 3 illustrates how these invariant distances can be summed into a $n \times n$ matrix that gives feature to feature distances between all n features and describes a global relationship between all the features in a scan.

n	f0	f1	f2	f3
f0	0	1	2	3
f1	1	0	3	4
f2	2	3	0	5
f3	3	4	5	0

(a) Scan $\mathbf{X} \in \mathbb{R}^{n \times n}$ feature to feature Geodesic distances

n	f?	f?	f?	f?
f?	0	3	2	1
f?	3	0	5	4
f?	2	5	0	3
f?	1	4	3	0

(b) Scan $\mathbf{Y} \in \mathbb{R}^{n \times n}$ feature to feature Geodesic distances

Fig. 3: Matrix 3a represents a global intrinsic model of the hand from Fig 1. The model is a list of geodesic distances between every pair of features; $x_{i,j} = |f_i - f_j|$ where: $i, j =< 1, \ldots, n >$. Matrix 3b is the geodesic distances between features as seen in Scan Y. The goal is to match the features from Scan Y to X using the known distance mappings in Matrix 3a, i.e., find the correct permutation of i, j for Matrix 3a to Matrix 3b s.t. $x_{i,j} = y_{i',j'} = |f_i - f_j|$

To register $\mathbf{X}$ and $\mathbf{Y}$ in Fig 3 the baseline constructs a MRF by applying a geodesic distance heuristic (GH) to each combination of $\mathbf{X}$ and $\mathbf{Y}$ and then uses a Tree Reweighted Message Passing (TRW-S) global optimizer to find a best matching. Since GH+TRW-S : $Y \mapsto X$, we refer to the baseline as "GH+TRW-S" to denote the registration function we accelerate. Although GH+TRW-S is global and requires no object specific priors, the trade off is it has greatly increased computation compared with the related work and online performance is very difficult to achieve.

The first compute heavy step is: *Geodesic Heuristic (**GH**)* where the registration is mapped as an energy minimization MRF problem by calculating a energy cost of matching pairs of points in X and Y. This energy cost function is a heuristic form of geodesic distance matching where a filtering and weighting scheme is applied to the geodesic distances to make the registration more robust to imperfections of 3D scanning systems [16]. The GH algorithm results in an $O(n^4)$ sized MRF which is illustrated in Fig 4.

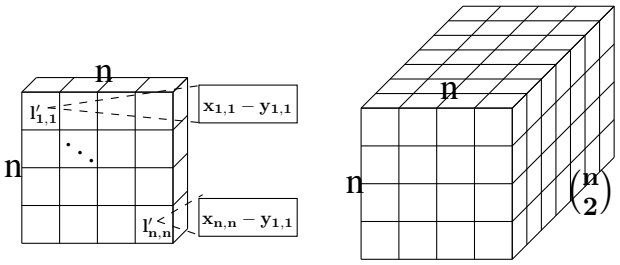
The second step is a *Global Optimal Matching* search using (***TRW-S***): TRW-S uses a global convex non-linear energy optimization to the explore the MRF problem and find a optimal matching candidate solution in $O(kn^4)$ time [21], where the registration solution calculates n matches and each scan has n features [16]. TRW-S is a popular MRF solver due to its fast convergence and its ability to compute a lower bound energy. It can report a theoretical best solution (the lower bound) which helps to choose a "k" constant for the overall $O(kn^4)$ complexity [22]. Full details on GH+TRW-S can be found in [16] and [21].

The major challenge to acclerating this algorithm is efficiently constructing $O(n^4)$ MRFs during the GH step, and then rapidly exploring this $O(n^4)$ problem space in the TRW-S phase. GH+TRW-S does not map well to most parallel architectures. In particular TRW-S has data dependencies that make it difficult to schedule.

TRW-S and other message passing based accelerators typically optimize DRAM streaming of the $\binom{n}{2}$ messages [17], [18], [2], [19]. However, we found GH+TRW-S to be memory bound when the MRF is ignored, since the TRW-S solver must stream in an MRF from the GH step as shown in Fig 2. The problem size does not need to be very large before the $O(n^4)$ MRF exceeds typical cache sizes and accelerators become bottlenecked. e.g., a 64 point registration requires 66 MB of cache [20].

A. Dynamic MRF Generation

Fig 2 shows that video rate performance is not possible using the baseline GH+TRW-S algorithm on any reviewed accelerator. 464GB/s bandwidth is required to achieve 20 scan/s video rate on a 110 node registration. Typically, hardware acceleration platforms have significantly less bandwidth. For example, our Arria 10 + Xeon hardware can only stream the MRF to a TRW-S accelerator at 12GB/s. To reduce this bandwidth requirement, our algorithm uses a novel approach called *dynamic Markov random field generation (DMRF)*.



(a) A MRF "Slice" (L') is the n^2 elements that make $\mathbf{X} - y_{i,j}$ (b) The complete MRF "Loaf" (L) is all $\binom{n}{2}$ Slices

Fig. 4: A MRF registration problem is constructed using two matrices X and Y of geodesic distances (see Fig 3). A good labeling is one where geodesic distances are the same ($\mathbf{X}_{a,b} - \mathbf{Y}_{i,j} = 0$). Therefore $\mathbf{X} - y_{i,j}$ are all n^2 potential matches for one $y \in Y$. This means checking all potential matches requires creating an MRF by subtracting all $y \in Y$ from X. We abstract this as a 3D tensor "loaf of bread": $L = [X - y_{1,1}, \ldots, X - y_{n,n}]$ of order $O(n^4)$.

Our DMRF algorithm dynamically computes the "slices" shown in Fig 4a as they are needed by TRW-S. DMRF is able to reduce MRF storage space because the number of independent random variables is actually far lower than n^4 for registration. Fig 3 shows only $2n^2$ variables are used to construct the $O(n^4)$ MRF for $\mathbf{X}$ and $\mathbf{Y}$ depicted in Fig 4b. DMRF dynamically evaluates the GH energy cost function:

$$E = (\exp(-\min(x/\sigma, y/\sigma)) \cdot \min(|x - y|, \tau)) \cdot q \quad (1)$$

for each slice in Fig 4b where σ and τ are GH filter and weight constants, respectively. q is a normalization factor for numerical stability [16]. Dynamic evaluation means only the $2n^2$ independent MRF variables are transferred to our accelerator. Fig 5 shows our platform is compute bound for larger problems with this approach. eg. 0.08GB/s is used for 20 scan/s 110 node DMRF transfer versus 464GB/s for MRF.

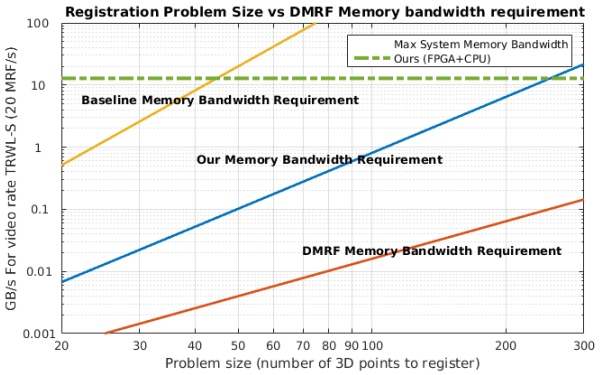


Fig. 5: DMRF removes the MRF transfer bottleneck ("Baseline" trend) which changes the problem from memory to compute bound ("DMRF" trend). However, scaling on our platform is now constrained by the "M" optimizer messages of Alg 1 ("Our" trend) and this is discussed in Sec V-D

Provided X and Y are available, Eqn 1 can be evaluated simultaneously for the entire MRF. This makes generating

loaf slices with DMRF an effective approach. Our algorithm integrating our DMRF generation into TRW-S optimization is described in Alg 1. We call this modified algorithm TRWL-S.

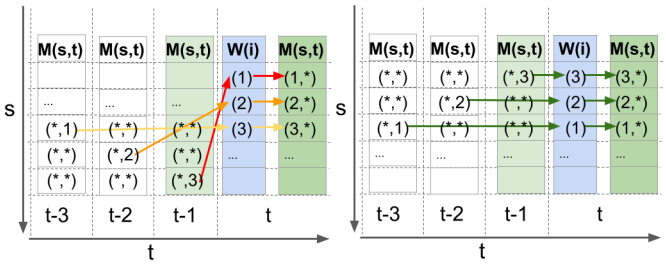
Algorithm 1 TRWL-S

procedure DMRF($\mathbf{X}, \mathbf{Y}, s, t, q$)
  $y = \mathbf{Y}$[s,t]
  return $E(\mathbf{X}, y).q$ ▷ see Eqn 1
procedure UD(M, L', W, s, t)
  off = $\Gamma[s,t] \circ W - M$ ▷ $\circ$ is Hadamard product†
  $G'[1:n, *] = L'[1:n, *]$+ off
  M' = COLMIN(G')
  return M' - MIN(M')
procedure TRWL-S($\mathbf{U}, \mathbf{X}, \mathbf{Y}, q, j$) ▷ $\mathbf{U}$ is optional*
  $\mathbf{M}[*,*] \leftarrow [0]$
  while j-=1 > 0 **do** ▷ j is number of passes
    $W_f \leftarrow W_b \leftarrow \mathbf{U}$
    for $s = n$; s>0; s-=1 **do** ▷ Back pass (TRW-S)
      for t=s-1; t>0; t-=1 **do**
        $W_b[t]$+= $\mathbf{M}[t, s]$
        $W_b[t]$+= $\mathbf{M}[s, t]$
      for t=s-1; t>0; t-=1 **do**
        $L' \leftarrow$ DMRF($\mathbf{X}, \mathbf{Y}, s, t, q$)
        $\mathbf{M}[t, s]$= UD($\mathbf{M}[t, s], L', W_b[t], s, t$)†
    for t=n; t>0; t-=1 **do** ▷ Front pass (ours)
      for s=0; s<t-1; s+=1 **do**
        $W_f[t]$+= $\mathbf{M}[t, s]$
        $W_f[t]$+= $\mathbf{M}[s, t]$
      for s=0; s<t-1; s+=1 **do**
        $L' \leftarrow$ DMRF($\mathbf{X^T}, \mathbf{Y}, s, t, q$)
        $\mathbf{M}[s, t]$= UD($\mathbf{M}[s, t], L', W_f[t], s, t$)†
    $S \leftarrow$SLN($\mathbf{M}, \mathbf{G}, \mathbf{U}$)†
  return S

*$\mathbf{U}$ *is an optional* ***TRW-S*** *seed and may be set to zero* [21]
†Γ, *message update (UD) and solution (SLN) detailed in* [21]

B. Scheduling Optimizations

TRWL-S must be scheduled carefully due to the data dependency on $W[t]$ in the UD() step. Alg 1 shows two scheduling options for the two optimization passes of TRWL-S "Back Pass" uses the naive TRW-S scheduling which is sub-optimal. The serial dependency issue is illustrated graphically in Fig 6a. Our "Front Pass" schedule in Alg 1 ensures that the first $\mathbf{M}[t]$ required to be added to a $W[t]$ in the next loop of the front pass is computed first and is shown in Fig 6b. This scheduling optimization is important for effectively speeding up TRWL-S. Data dependencies prevent a fully parallel schedule and so pipelining is a better execution scheduling option. Without our schedule, TRWL-S would suffer pipeline stalls of up to $n \times$ (UD() + MRF()) time. Since both of these functions are a $O(n^2)$ compute bottleneck, total stall time without our schedule is on the order of $O(n^3)$.



(a) Data dependencies with the Naieve TRW-S schedule in Alg 1 (b) Data dependencies with our schedule in Alg 1

Fig. 6: TRWL-S serial data dependencies between $\mathbf{M}$ and $\mathbf{W}$. s is the inner loop and is completed once before traversing along t. Fig 6a is the "Back Pass" TRW-S schedule in Alg 1 and Fig 6b is our "Front Pass" schedule in Alg 1. $\mathbf{M}[s, *]$ depends on $\mathbf{W}[s]$ from $t-1$ which in turn depends on $\mathbf{M}[*, s]$ from $t-2$. Our schedule in Fig 6b is better suited to pipelining as we schedule the compute of $\mathbf{M}$ to meet the dependencies of $\mathbf{W}$ at $t+1$ as early as possible.

C. MRF Optimization

The MRF cost function in Eqn 1 uses the exp function which is expensive in terms of both resource usage and latency. To improve this, we define a transformed cost function in Eqn 2:

$$E = \max(a, b) \cdot \min(|x - y|, \tau) \quad (2)$$

where $a = qe^{-x/\sigma}$ and $b = qe^{-y/\sigma}$. The transformation is valid because exp is a monotonic, increasing function of its argument, and $\max(-u, -v) = -\min(u, v)$ is an identity. The values of a and b are pre-computed on CPU using $O(n^2)$ work. We include the (positive) scaling term q in a and b when factoring out the expensive exp to reduce even further the total compute required for the $O(n^4)$ FPGA work .

D. Precision Optimization

In order to maximize the performance, we aim to saturate the FPGA multiplier blocks and other arithmetic resources. Key to this is using fixed point arithmetic since the target Arria 10 FPGA has more integer DSP multipliers than floating point multiplier blocks.

The GH+TRW-S baseline uses 64 bit floating point numbers. The Arria 10 DSP multipliers are 18 bits wide, so ideally we would prefer to reduce baseline precision to match the DSP multipliers. At the same time, in order to maintain accurate registration, the accelerator can not arbitrarily reduce the TRWL-S word width.

We analyzed the impact of reduced word width by sweeping fixed point precision of GH+TRW-S on our data sets to verify reduced precision did not negatively impact registration performance. Fig 7 shows that the energy minimization (registration capability) of GH+TRW-S is not greatly affected until we reduce the data representation to a 8 bit fixed point. We conservatively choose 16 bit fixed point since this precision maps well to the Arria 10 DSP multipliers, reduces memory footprint by a factor of 4X over GH+TRW-S, and

does not greatly impact the registration quality on our tested data sets. we do note that other data sets may vary although 16 bits is acceptable across our two tested data sets.

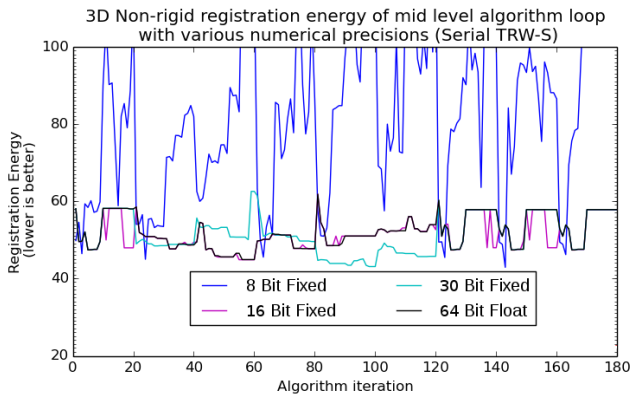


Fig. 7: The registration energy vs. the algorithm iteration for different data types. A lower energy indicates a more accurate registration. The 64 bit float, 30 bit fixed, and 16 bit fixed all have similar registration energy. The 8 bit fixed is significantly worse.

IV. ACCELERATOR ARCHITECTURE

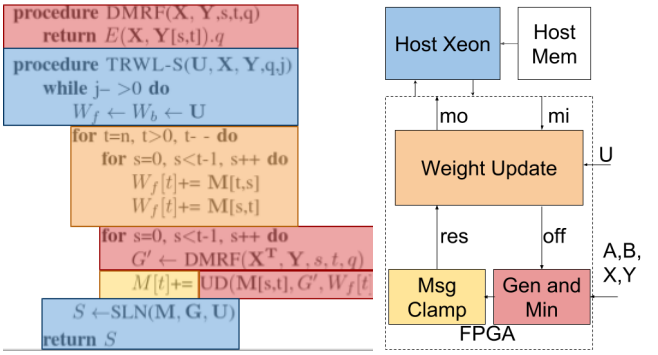
A. System Overview

Our accelerator is built using a 14 core Intel Xeon Broadwell coupled to a Altera Arria 10 FPGA via QPI and PCIe interconnects. The portions mapped to the Xeon were implemented in C++. The portions mapped to the FPGA were designed using Chisel3 [23]. The CPU to FPGA memory interface was built using Intel's *Rapid Design Methods for Developing Hardware Accelerators* methodology [24], which is optimized for CPU to FPGA data transfers of cache line granularity, i.e., 32 16 bit fixed precision numbers per FPGA cycle. Because of the CPU/FPGA memory interface, it is efficient to process data in vectors of 32 numbers and all block to block data transfers in Fig 8b use buffers 32 elements wide.

Fig 8 provides a high level block diagram of our heterogeneous non-rigid 3D registration accelerator. The Host Xeon provides high-level control, executes initialization procedures, and performs the mathematical transforms needed for registration. The functions DMRF() and UD() from Alg 1 are mapped to the FPGA since they can benefit from a parallel implementation. $\mathbf{W}$ operations from Alg 1 are also mapped to the FPGA. This is because the accelerator would suffer execution starvation from serial dependencies on $\mathbf{W}$ (see Fig 6) if the $\mathbf{W}$ variables had to be maintained by the CPU and streamed in as needed. The blocks in Fig 8b are each detailed in Sections IV-B to IV-E.

B. Gen and Min

"Gen and Min" in Fig 8b implements DMRF() and the COLMIN() of UD() in Alg 1 since these are the compute bottleneck of TRWL-S, both having $O(n^2)$ complexity per slice. Both DMRF() and COLMIN() are SIMD which our



(a) TRWL-S pass with hardware mapping highlighted

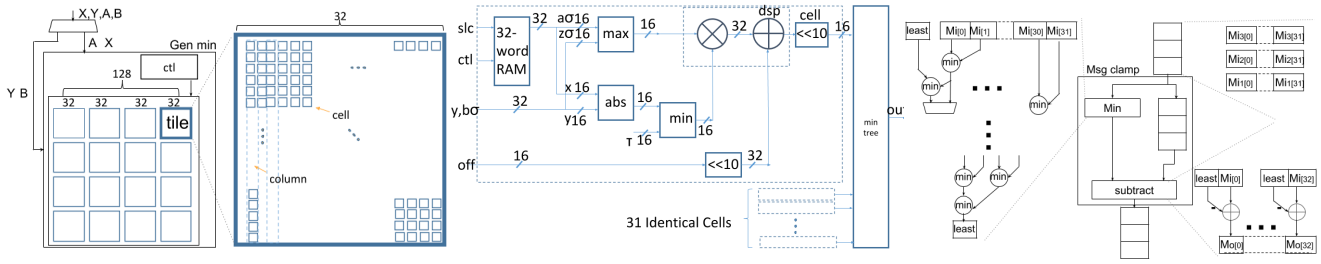
(b) Xeon/Arria 10 TRWL-S accelerator block diagram

Fig. 8: Heterogeneous TRWL-S accelerator block diagram. Fig 8a colours a TRWL-S pass to indicate how we partition the algorithm on our accelerator. Fig 8b is our Xeon/Arria 10 3D non-rigid registration accelerator. Each hardware block is coloured to match the portion of TRWL-S described in Alg 1 it is responsible for. We omit the Backward pass for brevity.

accelerator takes advantage of as shown in Fig 9. The block depicted in Fig 9a shows how larger matrices, for example, 128×128, are mapped to 4×4 virtual tiles using time-division multiplexing. Our architecture physically implements 32×32 compute cells. The tile arrangement can be modified at runtime to describe registration accelerators for different problem sizes (multiples of 32.) Eqn 2 shows that $\mathbf{A}$ and $\mathbf{X}$ are constant factors for every loaf slice and so our architecture loads the appropriate a and x into the cell ram shown in Fig 9b once only per registration. We use double buffering to allow tiles to begin work as $\mathbf{A}$ and $\mathbf{X}$ are loaded from CPU. Each cell implements Eqn 2 for DMRF() and adds "off" as the second step of UD() in Alg 1. Each cell's output is then fed to COLMIN() hardware which is shown as the min tree block in Fig 9b. Min tree performs the n wide vector min using a log tree arrangement with 32 word inputs. Processing a slice takes 16 cycles (one cycle for each tile.) The gen and min unit is deeply pipelined (14 stages) giving a L' slice per FPGA clock ratio of 1:1.

C. Message Clamp

Msg Clamp or "message clamp" in Fig 9c implements the final MIN() step of UD() in Alg 1. Using the separate message clamp block simplifies our architecture. It is broken out from "gen and min" (Section IV-B) because MIN() is an aggregating step that is not a compute bottleneck and is parallel in an orthogonal way to COLMIN() of the gen and min block. Msg clamp uses the same log tree vector min concept as gen and min but cycles the $least$ output several times through the tree. This is because the input from gen and min is a cache line (32 numbers) wide and so *least*= MIN(x) should be run once for every tile of gen and min. The FIFO in the clamper delays arrival of results from the first tile to a 32 wide vector subtract until a true MIN() has been computed. $\mathbf{M}$[.,.] is computed as the result of the vector subtract and streamed to the weight update block.

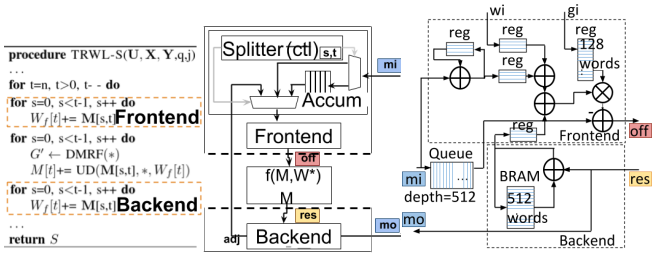


(a) Gen and Min tile and cell architecture (b) Cell and colmin block (c) Msg Clamp block

Fig. 9: Gen and Min block. This block is dedicated to the $O(n^2)$ per slice compute bottleneck of TRWL-S. It computes DMRF() loaf slices from Alg 1 and n vector mins of that slice. Vector mins are the bottleneck of UD() from Alg 1. We achieve high performance with an array of 32×32 cells each containing a 14 stage pipeline as shown in Fig 9a and detailed in Fig 9b.

D. Weight Update

"Weight Update" in Fig 10 maintains the $\mathbf{W}$ weight variables in Alg 1 because they are on the critical path to meet the serial dependencies of TRWL-S and round trip latency is too high to manage $\mathbf{W}$ on CPU. $\mathbf{W}$ is therefore stored on the FPGA in BRAM and is optionally initialized to $\mathbf{U}$ by the CPU. We re-order weight update as shown in Fig 10a so that the dependency of $t+1$ can be met by forwarding $\mathbf{M}[.,x] + \mathbf{W}[x]$. The forwarding stage is shown as the "Backend" in Fig 10a. We manage meeting the $\mathbf{W}$ dependencies of gen min (Sec IV-B) with the "Frontend" block in Fig 10a. This block tracks s and t, using these variables to route the correct message, $\mathbf{M}[.,.]$ and weight $\mathbf{W}[.]$. If the required dependency can't be satisfied, weight update will stall the accelerator. In practice stalls are negligible (0.2% stall cycles per MRF). If dependencies are met, $\mathbf{W}[.]$ and $\mathbf{M}[.,.]$ are scaled by Γ as shown in Fig 10b which is the first step of Alg 1 UD(). The result "off" is then passed to "gen and min".



(a) Partitioning and scheduling scheme of the Weight Update block (b) Data flow details of the Weight Update block

Fig. 10: Control and Data flow views of the Weight Update Block. Fig 10a shows a rescheduling of $\mathbf{W}$ computation that allows dependency forwarding of Alg 1 from t to $t+1$. Fig 10b details $\mathbf{W}$ processing. We use a local copy of $\mathbf{W}$ to efficiently compute dependencies. We perform canonical tree re-weighting with Γ in the "frontend" only for efficiency [21].

E. CPU

The CPU is responsible for control and data flow, making our 3D registration accelerator heterogeneous.

On the control path; The CPU starts TRWL-S and determines the number of iterations. We hard coded j in Alg 1 to be 20 iterations to match the GH+TRW-S base line for performance comparison. However, termination can also be determined dynamically by comparing the SLN() registration energy (solution quality) to either the lower bound [21] or an experimentally determined value.

The CPU can also optionally specify registration candidate solutions at each pass via the $\mathbf{U}$ parameter. We use the same $\mathbf{U}$ as the baseline for our performance evaluation.

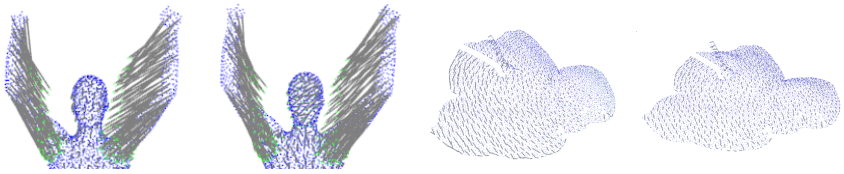
On the data path; The CPU pre-processes registration problems by computing $\mathbf{A}$ and $\mathbf{B}$ before starting TRWL-S as described in Section III-C. Other pre-processing steps include computing the baseline q normalization factor [16] and converting all numbers to 16bit Fixed point representation. Since the $\mathbf{M}$ messages require $O(n^3)$ storage, the CPU streams them off and on to the FPGA as needed. Finally the CPU decodes the 3D registration solution by implementing SLN() [21].

V. EVALUATION

We evaluate our system accelerator performance against a software implementation of the baseline GH+TRW-S and software TRWL-S. We compare only TRWL-S : $Y \mapsto X$ against GH+TRW-S : $Y \mapsto X$ since we do not re-implement the input stage (geodesic distance computation) or the output stage (expansion move upscaling). The baseline uses a state of the art implementation of TRW-S which has been enhanced with openMP to run a bottlenecking function (UD()) from Alg 1 in parallel when possible [20]. Our TRWL-S accelerator significantly outperforms the baseline implementation ($\approx$ 600X faster). This would make a holistic baseline software to accelerator comparison uninteresting. Therefore in this section we use the baseline to evaluate registration quality of the accelerator, but use a software TRWL-S implementation to compare other metrics. Specifically we evaluate accelerator run time performance and system power consumption of the accelerator against software TRWL-S. Finally we evaluate the scalability limitations of our TRWL-S non-rigid 3D registration accelerator.

Dataset Name	Scans	n	Mean qualityδ
Liver Surgery	25	96	1.3%
FAUST	50	96	1.9%
Liver Surgery	25	128	1.8%
FAUST	50	128	1.7%

(a) Registration δ from baseline (lower is better)



(b) Baseline: FAUST (c) Ours: FAUST (d) Baseline: Surgery (e) Ours: Surgery

Fig. 11: Summary of TRWL-S registration performance (Table 11a) and comparisons to GH+TRW-S (Figures 11b to 11e)

A. Registration Quality

We compared registration performance of TRWL-S to the baseline using the FAUST pose estimation database [25] and our own surgical video data set. Figure 11 shows the results. We calculate the mean difference of solution energy (registration quality) between baseline TRW-S and TRWL-S in Fig 11a accross our data sets. The average energy difference is small since the algorithms are equivalent except for the reduced numerical precision and registration results are therefore very similar. Figures 11b to 11e show a qualitative comparison of the 3D registrations of both algorithms on the two different data sets. In general TRWL-S appears to have good agreement with the baseline.

B. Accelerator Performance

We implemented the accelerator on a Xeon-FPGA platform using the Arria 10 10A115U3F45E2SGE3 FPGA. We used Chisel3 [23] and the *Rapid Design Methods For Developing Hardware Accelerators* [24] methodology for implementing the system. Chisel3 enables highly parameterized, clock accurate RTL descriptions to be developed in a concise and effective manner. We set the active matrix size to be 32×32 so that the *Gen and Min* component uses 1024 of the 1518 DSP blocks available on the FPGA. All components of the system together occupy 42% of the available ALMs (this includes both the Blue Stream, the interface to the Xeon provided by the platform, and the Green Stream, the hardware specific to this accelerator.) Timing converged to 400 MHz in the Blue Stream and 100 MHz in the Green Stream (no timing violations.) Other FPGAs with more compute resources (e.g., the Stratix 10 with up to 5760 DSP blocks) and higher clock rates (200 MHz or more can likely be achieved with more implementation work) would improve the performance proportionally.

C. System Power Consumption

We measured power consumption on the physical platform. The measurement system provides a breakdown by: CPU power, DRAM power, Core FPGA power, and Other FPGA power. For the software-only case, we measured power on the same system. Removing the FPGA components from the software-only case, we see a 69.6 W for software only vs. 72.8 W for the accelerated system. The runtime for the accelerated system is $\approx 84\times$ smaller (compared with a single core software-only implementation), resulting in an energy per computation benefit of $\approx 81\times$. The runtime and energy benefits are less when compared to a multi-threaded software implementation ($10\times$ and $15\times$, respectively) but still significant.

TABLE I: Measured Power for the Accelerator Computation

	CPU (W)		FPGA (W)		Perf per Frame	
Name	Core	DRAM	Core	Other	time(s)	energy(J)
FPGA Accel	24.2	28.1	8.7	11.8	0.05	3.6
SW (1 thrd)	41.5	28.1	(7.1)	(11.8)	4.22	294.0
SW (14 thrds)	86.0	28.1	(7.1)	(11.8)	0.50	55.2
Ratio (1)	1.71	1.00	0	0	84.5	80.8
Ratio (14)	3.55	1.00	0	0	10.0	15.2

"FPGA Accel" are power metrics for our non-rigid 3D registration TRWL-S accelerator system. "SW (1 thrd)/(14 thrds)" are metrics for software-only TRWL-S registration using 1 and 14 CPU hardware threads respectively. Adding threads makes a CPU compare increasingly favorably with FPGA, however the FPGA always maintains a large energy and speed advantage in our system

D. Scalability

The algorithm transformations described in Section III results in two major limits to scalability: 1) the $O(n^3)$ bandwidth requirement for transferring $\binom{n}{2}$ length n messages to and from memory, and 2) the $O(n^4)$ compute required to perform the *Gen and Min* operations. Figure 12 show both limitations on the same log-log graph.

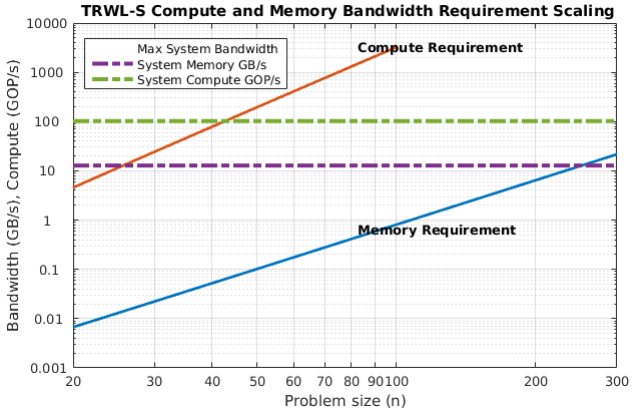


Fig. 12: TRWL-S scaling on a Xeon Aria 10 platform. Memory bandwidth scaling is dominated by message streaming which is $O(n^3)$. Compute scaling is dominated by the $O(n^4)$ DMRF computation. Compute must improve by 17x (using a faster clock or larger FPGA) before platform bandwidth is the performance limiter (at n=256).

Compute can be increased by clocking the system faster or utilizing an FPGA with more resources (DSP blocks). The horizontal line showing the compute limit would increase proportionally. It can increase 17x before the platform bandwidth becomes the bottleneck around $n = 256$.

VI. CONCLUSIONS

We describe a heterogeneous CPU/FPGA accelerator for real-time non-rigid 3D registration. The design uses a MRF transform and scheduling optimizations to achieve 20 registrations per second performance. Experimental results show clear performance benefits of the accelerator. Our system achieves $\approx 600\times$ speed up with a maximum 1.9% difference in registration quality over a software only TRW-S non-rigid 3D registration baseline. Additionally we find a $\approx 84\times$ speed improvement and $\approx 81\times$ energy reduction of our heterogeneous TRWL-S architecture versus a software only implementation. In future work we plan to further reduce the 1.9% quality difference by tuning the registration parameters of TRWL-S and to investigate current scaling limitations with our approach.

ACKNOWLEDGMENT

The authors gratefully acknowledge that this work was supported in part by:
- *Cognex Corporation*
- *The ACM SIGHPC/Intel Computational & Data Science Fellowship*

REFERENCES

[1] R. A. Newcombe, D. Fox, and S. M. Seitz, "Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 343–352.

[2] J. Choi and R. A. Rutenbar, "Video-rate stereo matching using markov random field trw-s inference on a hybrid cpu+ fpga computing platform," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 2, pp. 385–398, 2016.

[3] J. Stückler and S. Behnke, "Multi-resolution surfel maps for efficient dense 3d modeling and tracking," *Journal of Visual Communication and Image Representation*, vol. 25, no. 1, pp. 137–147, 2014.

[4] A. Hughes-Hallett, E. K. Mayer, P. Pratt, A. Mottrie, A. Darzi, and J. Vale, "The current and future use of imaging in urological robotic surgery: a survey of the european association of robotic urological surgeons," *The International Journal of Medical Robotics and Computer Assisted Surgery*, vol. 11, no. 1, pp. 8–14, 2015.

[5] P. Pessaux, M. Diana, L. Soler, T. Piardi, D. Mutter, and J. Marescaux, "Towards cybernetic surgery: robotic and augmented reality-assisted liver segmentectomy," *Langenbeck's archives of surgery*, vol. 400, no. 3, pp. 381–385, 2015.

[6] S. Ieiri, M. Uemura, K. Konishi, R. Souzaki, Y. Nagao, N. Tsutsumi, T. Akahoshi, K. Ohuchida, T. Ohdaira, M. Tomikawa *et al.*, "Augmented reality navigation system for laparoscopic splenectomy in children based on preoperative ct image using optical tracking device," *Pediatric surgery international*, vol. 28, no. 4, pp. 341–346, 2012.

[7] A. Schoob, D. Kundrat, L. Kleingrothe, L. A. Kahrs, N. Andreff, and T. Ortmaier, "Tissue surface information for intraoperative incision planning and focus adjustment in laser surgery," *International journal of computer assisted radiology and surgery*, vol. 10, no. 2, pp. 171–181, 2015.

[8] M. Ye, H. Wang, N. Deng, X. Yang, and R. Yang, "Real-time human pose and shape estimation for virtual try-on using a single commodity depth camera." *IEEE transactions on visualization and computer graphics*, vol. 20, no. 4, pp. 550–559, 2014.

[9] M. Dou, S. Khamis, Y. Degtyarev, P. Davidson, S. R. Fanello, A. Kowdle, S. O. Escolano, C. Rhemann, D. Kim, J. Taylor *et al.*, "Fusion4d: Real-time performance capture of challenging scenes," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 114, 2016.

[10] R. Madhavan, T. Hong, and E. Messina, "Temporal range registration for unmanned ground and aerial vehicles," *Journal of Intelligent and Robotic Systems*, vol. 44, no. 1, pp. 47–69, 2005.

[11] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. Ieee, 2011, pp. 1297–1304.

[12] C. Keskin, F. Kıraç, Y. E. Kara, and L. Akarun, "Real time hand pose estimation using depth sensors," in *Consumer depth cameras for computer vision*. Springer, 2013, pp. 119–137.

[13] M. Siddiqui and G. Medioni, "Human pose estimation from a single view point, real-time range sensor," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*. IEEE, 2010, pp. 1–8.

[14] M. Zollhöfer, M. Nießner, S. Izadi, C. Rehmann, C. Zach, M. Fisher, C. Wu, A. Fitzgibbon, C. Loop, C. Theobalt *et al.*, "Real-time non-rigid reconstruction using an rgb-d camera," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 156, 2014.

[15] R. W. Sumner, J. Schmid, and M. Pauly, "Embedded deformation for shape manipulation," in *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3. ACM, 2007, p. 80.

[16] Q. Chen and V. Koltun, "Robust nonrigid registration by convex optimization," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2039–2047.

[17] S. Hurkat, J. Choi, E. Nurvitadhi, J. F. Martínez, and R. A. Rutenbar, "Fast hierarchical implementation of sequential tree-reweighted belief propagation for probabilistic inference," in *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*. IEEE, 2015, pp. 1–8.

[18] C.-K. Liang, C.-C. Cheng, Y.-C. Lai, L.-G. Chen, and H. H. Chen, "Hardware-efficient belief propagation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 5, pp. 525–537, 2011.

[19] W. Zhao, H. Fu, G. Yang, and W. Luk, "Patra: Parallel tree-reweighted message passing architecture," in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*. IEEE, 2014, pp. 1–6.

[20] "Robust nonrigid registration by convex optimization: Complete code," "https://drive.google.com/file/d/0B3j4PjSVMIHLYWttRks5N0tmTXM/view?usp=sharing, accessed: 2017-03-30.

[21] V. Kolmogorov, "Convergent tree-reweighted message passing for energy minimization," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 10, pp. 1568–1583, 2006.

[22] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother, "A comparative study of energy minimization methods for markov random fields with smoothness-based priors," *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 6, pp. 1068–1080, 2008.

[23] "Chisel3," "https://github.com/freechipsproject/chisel3.git, accessed: 2017-03-30.

[24] "Rapid design methods for developing hardware accelerators," "https://github.com/intel/rapid-design-methods-for-developming-hardware-accelerators.git, accessed: 2017-03-30.

[25] F. Bogo, J. Romero, M. Loper, and M. J. Black, "Faust: Dataset and evaluation for 3d mesh registration," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3794–3801.