The Parallel Iterative Closest Point Algorithm

Christian Langis Michael Greenspan Guy Godin

Visual Information Technology Group
Institute for Information Technology, National Research Council Canada
Bldg. M-50, 1500 Montreal Rd., Ottawa, Ontario, Canada, K1A 0R6
email: *firstname.lastname*@nrc.ca

Abstract

This paper describes a parallel implementation developed to improve the time performance of the Iterative Closest Point Algorithm. Within each iteration, the correspondence calculations are distributed among the processor resources. At the end of each iteration, the results of the correspondence determination are communicated back to a central processor and the current transformation is calculated. A number of additional techniques were developed that served to improve upon this basic scheme. Calculating the partial sums within each distributed resource made it unnecessary to transmit the correspondence values back to the central processor, which reduced the communication overhead, and improved time performance. Randomly distributing the points among the processor resources resulted in a better load balancing, which further improved time performance. We also found that thinning the image by randomly removing a certain percentage of the points did not improve the performance, when viewed as the progression of mse with time. The method was implemented and tested on a 22 node Beowulf class cluster. For a large image, linear performance improvements were obtained for up to 16 processors, while they held for up to 8 processors with a smaller image.

1 Introduction

The Iterative Closest Point Algorithm (ICP) has become established as one the most useful methods of range data processing. Given two sets of partially overlapping range data and an initial estimate of their relative positions, ICP is used to *register* the data sets by improving the position and orientation estimate. First described by Besl and McKay [1], ICP is an essential step in model building, dimensional inspection, and numerous applications of range data processing.

At each ICP iteration, correspondences are determined between the two data sets, and a transformation is computed which minimizes the mean square error (mse) of the correspondences. The iterations continue until either the mse falls below some threshold value, the maximum number of iterations is exceeded, or some other stopping condition is satisfied. Due to its fairly large computational expense, ICP is typically considered to be a batch or, at best, a user-guided online process where the user initiates and assists the process and then allows it to execute unsupervised for a number of minutes. An ICP which executes in real-time, or near real-time, would prove advantageous in several situations. One reason is that range data acquisition sensors are getting faster. The combination of real-time range data acquisition with a real-time ICP forms the basis of a number of new and useful systems, such as geometric tracking and hand-held sensors. Furthermore, there are emerging applications such as environment modeling where the volume of data is large, and for which any improvement in the speed of ICP is desirable.

There has been some previous work in developing efficient versions of ICP. Simon et al. [2] implemented efficient correspondence calculations based upon the k-d tree and decoupled acceleration for the rotation and translation components. Jasiobedzki et al. [3] implemented and compared a variety of model representations and correspondence methods. Greenspan and Godin [4] proposed a correspondence method which is specifically tailored to efficient ICP.

A generic method for speeding up computations is *parallelization*. There has been a recent resurgence in the popularity of parallel computing, which can be attributed to the improved availability, ease of use, cost effectiveness, and power of PC clusters. Images are an inherently parallel data structure, and image processing methods are likely to benefit directly from the proliferation of parallel computing systems. In this work we describe a parallelized version of the ICP which we call $pICP$. The parallelization

scheme proposed here is coarse grain, in that tens of thousands of operations or more are executed by each parallel resource between any synchronization or communication. But it is likely that some of the techniques that we present here also apply to the fine grain case.

The objective of parallelization is to approach linear improvement, i.e. if it takes time t to execute using a single processor, we would like it to take only time t/N for N processors. While some methods are mired in the sequential framework and seem not to be amenable to parallelization, others parallelize trivially. If a method is parallelizable, then there are two general design principles which must be followed to maximize speedup. The first is to reduce the ratio of communications to computation. The second is to balance the computational load equally among the resources.

This paper continues with a description of the basic $pICP$ method in Section 2. In Section 3, a number of techniques are described which aim at improving upon the basic $pICP$, including parallelized computation of cross-covariances, shuffling, thinning, and acceleration. A set of experiments are reported in Section 4 which characterize the performance of the method. The paper concludes in Section 5 with a summary and a discussion of future research.

2 Basic method

The ICP algorithm iteratively registers a *floating* (i.e. *data*) surface towards a *reference* (i.e. *model*) surface. In the original description of the method [1], the surfaces can be described by a number of representations, such as point sets, line segments, triangle sets, etc. The only requirement is the existence of a closest point operation which returns, for a given query point, the point on the reference surface that is closest to the query point.

In this work, we consider the case where both surfaces are described as 3D point sets. The decision to restrict ourselves to this geometric representation was motivated by the objective of developing a process which performs registration between range images in real-time as they are acquired. Nevertheless, the parallelization techniques presented here generalize directly to the use of other surface representations, and some other registration methods.

Let the floating and reference images be respectively denoted as I_f and I_r. The basic ICP algorithm is well known [1], but we restate it briefly here for completeness:

1. *Establish correspondences:* the closest point on the surface of I_r is sought for all points in I_f.
2. *Derive transformation:* the correspondences are used to compute a transformation matrix which minimizes the mean square error (mse) of the residuals.
3. *Transform I_f:* the transformation is applied to all points of I_f.

```
// Parent

// initialize
IterationCount = 0
I_{f_{1..N}} = PartitionImage(I_f)
for(i = 1..N)
{   SendImagePartition(Child_i, I_{f_i})
    SendImage(Child_i, I_r)
}

// iterate
while (stopping condition not satisfied)
{   for(i = 1..N)
    {   Correspondences[i]
            = ReceiveCorrespondence(Child_i)
    }
    Transform = CalcTransform(Correspondences)
    for(i = 1..N)
    {   SendTransform(Child_i, Transform) }
    ApplyTransform(Transform, I_f)
    MSE = CalcError(I_f, I_r)
    IterationCount++
}
stop all Children
return I_f


// Child

// initialize
I_{f_i} = ReceiveImagePartition(Parent)
I_r = ReceiveImage(Parent)
BuildElias(I_r)

// iterate
while (TRUE)
{
    Correspondences = CalcCorrespondences(I_{f_i}, I_r)
    SendCorrespondences(Parent, Correspondences)
    Transform= ReceiveTransform(Parent)
    ApplyTransform(Transform, I_{f_i})
}
```

Figure 1: Pseudocode of basic $pICP$

4. *Terminate:* if the change in mse falls below a preset threshold, then terminate. Else, repeat from step 1.

The mse metric which is used to evaluate the goodness of the registration is the average of the squared distances between corresponding points, i.e. the average of the square of the residuals. It has been shown that if the transformation derived in step 2 minimizes the mse of the residuals, then the mse is bound to reduce monotonically over the iterations. The iterative structure of the algorithm imposes that any parallelization must take place within each iteration.

Within an iteration, establishing the correspondences accounts for the largest portion of the processing expense, and is the most likely candidate for parallelization. Efficiently determining the closest point from a set of possi-

bilities is a classical problem of Computational Geometry called the *Nearest Neighbor Problem*. A variety of practical solution methods exist, such as k-d tree [5] or Elias (or voxel-binning) methods [6], which greatly reduce the cost of establishing correspondences over exhaustive searching. In our tests, we have used a straightforward implementation of the Elias which we found to be very time efficient when compared to the k-d tree method [4]. Regardless of the method used, a nearest neighbor must be found for every n_f points of the floating image I_f. Hence, the most natural strategy parallelizes the correspondence step by partitioning I_f into subsets across the N processes.

2.1 Parallelization

A pseudocode representation of the basic parallelized version of the method is depicted in Fig. 1. At initialization, the complete reference image I_r is sent to each of the N *child* processes. The *parent* process partitions I_f into N disjoint subsets I_{f_i}, and transmits each subset to a child process. At each iteration, each child process establishes the correspondences between its I_{f_i} and I_r, and sends the correspondences back to the parent. The parent then calculates the incremental transformation and sends it to the children, which apply it to their I_{f_i} in preparation for the next iteration. The basic $pICP$ is composed of the following operations:

Data partition: The parent process partitions I_f into N disjoint subsets. Since the correspondence search time of each subset increases linearly with the number of points in the subset, dividing I_f into subsets I_{f_i} of approximately equal cardinality should distribute the workload almost evenly across the child processes.

Initial data transmission: The parent transmits a subset I_{f_i} to each child process. Each child will work exclusively on this subset throughout the registration. Although this initial communication is intensive, it fortunately occurs only once every $pICP$ call.

Parallel iterations: Each child concurrently computes the correspondences between points in I_{f_i} and the points of I_r. The resulting correspondences are then sent back to the parent process.

Sequential parent computations: Once it receives all correspondences from the child processes, the parent then derives a transformation as in the sequential ICP. At little additional expense, the parent also applies the transformation to its copy of I_f and computes the mse.

Transformation redistribution: The parent process then transmits the transformation back to the children. The children apply the received transformation to their I_{f_i}, thus keeping the distributed copy of I_f consistent with the parent's copy. The parent will stop the iterative process whenever the requested registration accuracy, or some other condition such as a maximum number of iterations, has been met.

3 Enhancements to the parallel method

At this point, the $pICP$ runtime T behaves linearly in terms of N (the number of child processes) up to a certain limit of processes, which is characteristic of parallel applications. As N increases, the relative cost of communications becomes dominant. Also, the time limiting factor for an iteration is always determined by the slowest process. Thus the basic $pICP$ can be enhanced by reducing the amount of interprocess communications and by load balancing. A further approach to performance enhancements is to reduce the total amount of computations using subsampling of the data and parametric acceleration. The particulars of each of these solutions, along with their relative impact, are presented here.

3.1 Quaternion method

In the simple method outlined in Figure 1, the child processes must transmit at each iteration the list of correspondences for their subset to the parent. Derivation of the transformation requires the traversal of the n_f correspondence pairs to compute their centroids as well as a 4×4 matrix $\mathbf{M}$ from terms of the cross-covariance matrix of the pairings [7].

Explicit transmission of the correspondence information results in considerable data traffic and introduces a significant time expense per iteration. Furthermore, sequential addition of the residuals on the parent node does not make use of the available parallel processing power.

A first improvement to the basic parallelization distributes the calculation of the cross-covariance terms across the children. After calculating the local correspondences for their subsets of I_f, each child process computes partial sums, which are then sent to the parent and combined into the total cross-covariance matrix.

This modification of the method has three positive effects on performance. First, it significantly reduces the amount of data which must be exchanged between the children and the parent at each iteration: the basic scheme required the transmission of a total of n_f integers (the indices of the corresponding points) at each iteration, whereas the proposed modification requires only $16N$ values. Secondly, the amount of communication per iteration becomes independent of the size of the images being registered. Thirdly, the degree of parallelism is increased by dividing the burden of the computation of cross-covariance terms among the child processes.

For notation simplicity, the modification to the basic method is described here in general terms for a set of N_t pairings $(\mathbf{p}_i, \mathbf{q}_i)$, with points represented as column vectors of dimension 3×1. With the ICP method, these pairs are those found between I_f and I_r at each iteration.

We are seeking the rigid transformation that minimizes $\sum_{i=1}^{N_t} \|\mathbf{p}_i - \mathbf{q}_i\|^2$. It is computed using the quaternion-based least squares method [7], which requires computation of the centroids and cross-covariance matrix of the two point sets, from which matrix $\mathbf{M}$ is computed. The centroids of the paired points are defined as:

$$\bar{\mathbf{p}} = \frac{1}{N_t}\sum_{i=1}^{N_t}\mathbf{p}_i \quad , \quad \bar{\mathbf{q}} = \frac{1}{N_t}\sum_{i=1}^{N_t}\mathbf{q}_i \tag{1}$$

respectively, and the cross-covariance matrix $\mathbf{S}$ as:

$$\begin{aligned}\mathbf{S} &= \frac{1}{N_t}\sum_{i=1}^{N_t}(\mathbf{p}_i - \bar{\mathbf{p}})(\mathbf{q}_i - \bar{\mathbf{q}})^T \\ &= \Big(\frac{1}{N_t}\sum_{i=1}^{N_t}\mathbf{p}_i\mathbf{q}_i^T\Big) - \bar{\mathbf{p}}\bar{\mathbf{q}}^T \end{aligned} \tag{2}$$

In the parallelized scheme, the set of pairings is broken into N_p disjoint subsets, each of cardinality N_i, with pairings noted as $(\mathbf{p}_{ij}, \mathbf{q}_{ij})$. As well, the centroids for each subset are defined as $\bar{\mathbf{p}}_i$ and $\bar{\mathbf{q}}_i$. The sum of Eq. 2 can be broken into partial sums $\mathbf{C}_i$; the computation of $\mathbf{S}$ can then be rewritten as:

$$\mathbf{C}_i = \sum_{j=1}^{N_i}\mathbf{p}_{ij}\mathbf{q}_{ij}^T \quad , \quad \mathbf{S} = \frac{1}{N_t}\Big(\sum_{i=1}^{N_p}\mathbf{C}_i\Big) - \bar{\mathbf{p}}\bar{\mathbf{q}}^T \tag{3}$$

where the centroids $\bar{\mathbf{p}}$ and $\bar{\mathbf{q}}$ are computed as:

$$\bar{\mathbf{p}} = \frac{1}{N_t}\sum_{i=1}^{N_p}N_i\bar{\mathbf{p}}_i \quad , \quad \bar{\mathbf{q}} = \frac{1}{N_t}\sum_{i=1}^{N_p}N_i\bar{\mathbf{q}}_i \tag{4}$$

Each child reports 16 values to the parent: the 3×3 second-order elements of $\mathbf{C}_i$, 2×3 centroid elements, and N_i, the number of pairs in the subset. In the specific case of the $pICP$, if all query points in a subset produce a pairing, then only 12 values per child need to be transmitted, since N_i remains constant, and the updated centroids of the query points in each subset are readily available to the parent node.

Although the expression in Eq. 2 is algebraically correct, it may be subject to numerical errors for large numbers of points when the centroid is far from the origin, due to the subtraction of two potentially large terms. The following alternative version avoids this potential difficulty at a minimal additional computational cost on the parent processor:

$$\begin{aligned}\mathbf{S} &= \frac{1}{N_t}\sum_{i=1}^{N_t}(\mathbf{p}_i - \bar{\mathbf{p}})(\mathbf{q}_i - \bar{\mathbf{q}})^T \\ &= \frac{1}{N_t}\sum_{i=1}^{N_p}\sum_{j=1}^{N_i}(\mathbf{p}_{ij} - \bar{\mathbf{p}})(\mathbf{q}_{ij} - \bar{\mathbf{q}})^T \\ &= \frac{1}{N_t}\sum_{i=1}^{N_p}\sum_{j=1}^{N_i}\Big((\mathbf{p}_{ij} - \bar{\mathbf{p}}_i + (\bar{\mathbf{p}}_i - \bar{\mathbf{p}})) \\ &\qquad (\mathbf{q}_{ij} - \bar{\mathbf{q}}_i + (\bar{\mathbf{q}}_i - \bar{\mathbf{q}}))^T\Big) \\ &= \frac{1}{N_t}\sum_{i=1}^{N_p}\Big(\mathbf{S}_i + N_i(\bar{\mathbf{p}}_i - \bar{\mathbf{p}})(\bar{\mathbf{q}}_i - \bar{\mathbf{q}})^T\Big), \end{aligned} \tag{5}$$

$$\text{where } \mathbf{S}_i = \sum_{j=1}^{N_i}(\mathbf{p}_{ij} - \bar{\mathbf{p}}_i)(\mathbf{q}_{ij} - \bar{\mathbf{q}}_i)^T \tag{6}$$

The global centroids $\bar{\mathbf{p}}, \bar{\mathbf{q}}$ are obtained from $\bar{\mathbf{p}}_i, \bar{\mathbf{q}}_i$ using Eqs. 4.

In this improved method, almost all computations are parallelized. After finding the N_i pairs, each child computes the centroids $\bar{\mathbf{p}}_i$ and $\bar{\mathbf{q}}_i$. Then, the local cross-covariance matrix $\mathbf{S}_i$ (that is, relative to $\bar{\mathbf{p}}_i$ and $\bar{\mathbf{q}}_i$) is computed (Eq. 6). Each child sends $\bar{\mathbf{p}}_i, \bar{\mathbf{q}}_i, \mathbf{S}_i, N_i$ to the parent, which combines them sequentially using Eq. 5. The rigid transformation for all N_t pairs can finally be estimated with the method in [7].

This simple optimization brought a tangible difference in the execution times of our algorithm. Relieved from heavy communications at every iteration, the algorithm shows significant speedup throughout iterations. The difference in time between the parallelized and the sequential computations methods is shown for a typical run in Fig. 2.

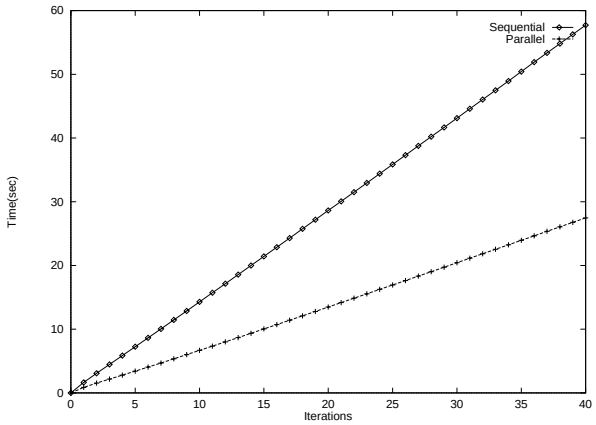


Figure 2: Sequential vs. parallel computation of cross-covariances.

3.2 Shuffling

At initialization of the $pICP$, the floating image I_f is divided into N subsets I_{f_i}, one for each child process. The subsets are of equal size, in an attempt to balance the workload evenly across all children. This balancing is crucial to the global performance of the $pICP$, since one iteration

cannot start before all the children have completed the previous one. Were the correspondence computation a constant time operation, then any equal division of the points between processes would indeed achieve the goal. However, this is not the case in general.

The correspondence computation is a Nearest Neighbor problem, which is a classical problem of Computational Geometry for which a variety of solution methods exist, such as k-d tree and Elias methods (e.g. [5, 6]). A common property of most of these methods is that they tend to execute with increasing efficiency as the actual distance to the nearest neighbor decreases. The calculation will therefore execute more efficiently for regions of I_f for which the residuals to I_r are smaller on average.

The images used here were acquired using a *Biris* laser range sensor [8] and are ordered as 2D arrays of 3D points, as is typical of imaging range sensors. Thus, points stored contiguously in the image array tend to be located in the same neighborhood of the measured surface.

In our initial implementation, the subsets of I_f were derived by simply dividing the image into contiguous blocks. Thus, points $(i-1)(\frac{n_f}{N})+1,...i(\frac{n_f}{N})$ would be assigned to child i. Under this scheme, some children were assigned regions of I_f which were either very well or very poorly registered with I_r, leading to an unbalanced workload. Since the time for an iteration of the $pICP$ is bound by the completion of the slowest of the N processes, this issue had to be addressed.

If the points of I_f were uniformly randomly ordered then all subsets of I_f would have approximately the same average residual size for a given iteration, and the correspondence computation would bear approximately the same cost across subsets. A simple process known as *shuffling* was implemented in an attempt to distribute the workload more evenly at no additional cost. With shuffling, child i is assigned points $i, N+i, 2N+i, ...(\frac{n_f}{N}-1)N+i$. Each subset is therefore composed of a set of points uniformly dispersed across I_f.

The improvement in the distribution of the workload using shuffling is illustrated in Fig. 3. The $pICP$ was run on the same data set for the same starting conditions with and without shuffling. The graphs show the execution time per iteration for 11 child processes. Clearly in Fig. 3 b) shuffling reduces the variation in execution time between processes, and the curves are tightly grouped together. Significantly, after a few iterations, the total time was reduced by $\approx 15\%$, from 3.2 to 2.7 seconds, when shuffling was applied. The gain is even larger during the first few iterations where the curves exhibit more variability. Shuffling also tends to yield similar centroids of pairings for each child process, which makes Eq. 5 even more numerically stable.

However, the experimentations with shuffling did not always yield positive results. One of the undeniable benefits of shuffling is to group together the individual child timings rather than leaving the original point ordering of I_f determining the variance of workload on the children. However, sometimes, there was still no speedup; the slowest child with shuffling was not any faster than without shuffling.

An interesting behavior was observed when evaluating the aggregate time (sum of all of the 11 individual child times) for each iteration. The aggregate times of the runs with shuffling exceeded those without shuffling by 20%. Our analysis attributed this unexpected performance to a poorer data cache usage. It should be remembered that the aggregate time is not in itself the measure of performance for a parallel algorithm. It does, however, provide the means for a comparison of the overhead of parallelism between different trials.

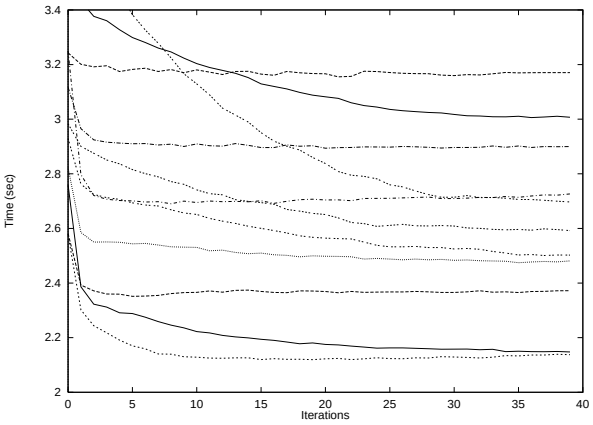
Without shuffling, the correspondences for each subset of I_f are rather well localized on a specific region of I_r. However, when the subsets are uniform samples of I_f, their individual correspondences span I_r completely. Hence, the consequence of uniformly distributed subsets is that each child must probe I_r completely, as opposed to a specific region of I_r in the non-shuffled case. And since the Elias data structure for typical I_r images rarely fits completely into the available cache memory, shuffling often causes much lower cache hit rates than the non-shuffled case. For this reason, shuffling must be used with caution, adjusted to the amount of cache available to each child CPU. Modifications of some key data structures used in the Nearest Neighbor computation are under way to alleviate this effect.

3.3 Thinning

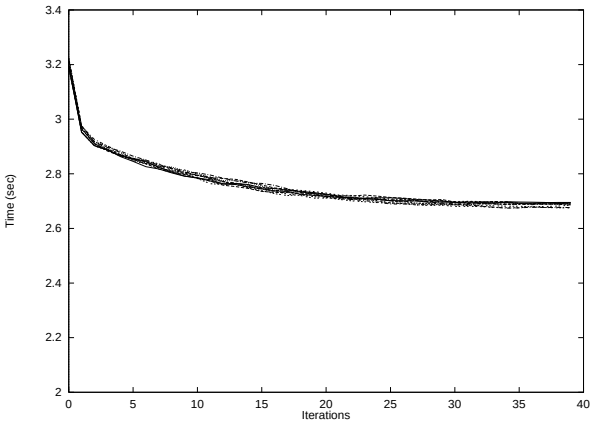
Using either the sequential or parallel method, the time to establish correspondences increases linearly with the size of I_f. One is therefore tempted to speed up ICP (and $pICP$) by using only a smaller portion of the points in I_f. Masuda and Yokoya [9] used random sampling to remove outliers, and the technique of subsampling has become a common practice to improve computational efficiency.

We evaluated the effect of reducing the size of I_f with a technique called *thinning*. At the beginning of each iteration, the processes randomly subsample their I_{f_i} subsets, at a factor $R \in (0, 1]$. Following the subsampling, only the $R\frac{n_f}{N}$ points selected by the subsampling filter are considered further in the correspondence computations.

To study the effect of thinning, experiments were performed using the sequential ICP (i.e. only one child process) for six values of R. Figure 4 plots the cumulative time per run as a function of iterations. As expected, the time per iteration decreases linearly with the value of R. In Fig. 5, the mse is plotted as a function of iterations.



a) without shuffling



b) with shuffling

Figure 3: Effect of shuffling

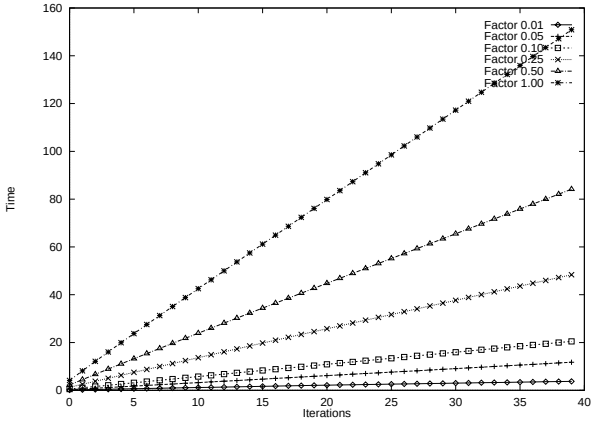


Figure 4: Thinning: time vs. iterations

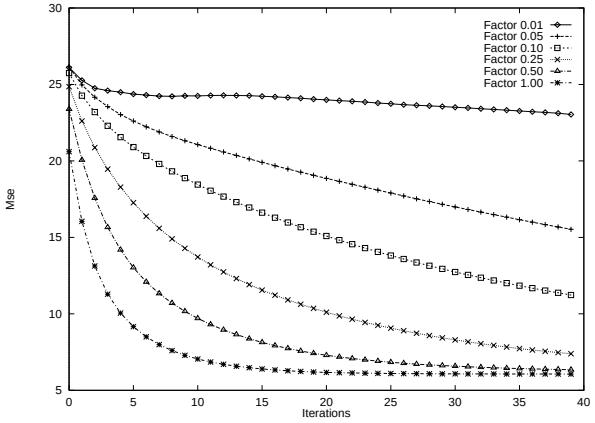


Figure 5: Thinning: mse vs. iterations

Despite the random subsampling of I_f at the beginning of each iteration, it can be seen that for a given iteration the mse value is smaller for the more densely sampled subsets.

An interesting conclusion can be drawn from Fig. 6, where the evolution of the mse with time is seen to be nearly identical for all values of R. By decreasing the value of R, the slower convergence of the mse almost exactly cancels out the faster iterations. From this graph, it appears that there is no efficiency improvement to be gained by thinning the image.

3.4 Acceleration

Another enhancement which is not specific to the parallel implementation is parametric acceleration [1]. It is a technique that can be applied on top of ICP to quicken the convergence: it does not speedup the execution time of the iterations, but instead skips some of them by applying an extrapolation factor based on the 6D trajectory of the transformation.

Estimation of the acceleration factor does not require correspondence computations; its execution time is insignificant compared to that of a single iteration. Therefore, it seems appropriate to take advantage of this technique as much as possible. However, the transforms it derives do not follow the ICP monotonic convergence property. In fact, our particular implementation of acceleration showed that very few accelerated transforms yielded smaller mse. Nevertheless, this does not prevent its use since a disadvantageous acceleration step can simply be discarded.

The best result is shown in Fig. 7. The acceleration parameters were especially tuned for this data set (for maximum acceleration). The two curves show the mse convergence for accelerated and non-accelerated ICP runs. Both curves appear superposed until iteration 4, where the acceleration kicks in and drives the mse lower on the accelerated curve. This was the only beneficial acceleration for this registration run. Furthermore the non-accelerated curve catches up quickly with the accelerated one in the

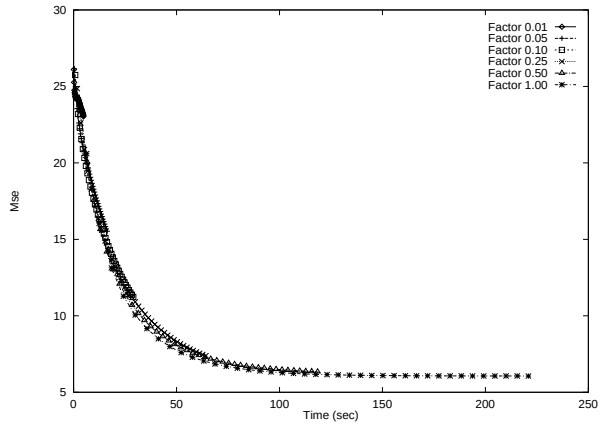


Figure 6: Thinning: mse vs. Time

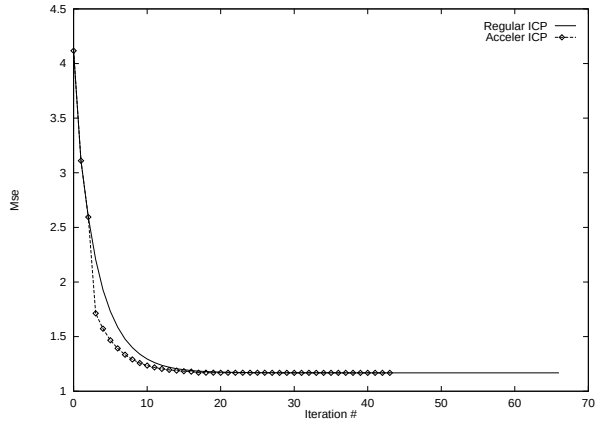


Figure 7: Acceleration

region of the figure where mse convergence slows down heavily. In this figure, acceleration eliminated about 20 iterations out of over 60. However, on average with different data sets less than 10% of iterations are spared which is far less performance than what we expected out of the technique.

Further experimentations were also conducted with another acceleration method [2]. This method, based on the previous one, breaks up the transformation into rotation and translation, each with a different extrapolation factor. Again, we observed no tangible improvements.

Although acceleration makes sense in theory, experimentations showed that a transform slightly off the transform trajectory generated with regular iterations causes evident regressions in terms of mse. Hence such a transform close to the transformation path, might possibly generate a much higher mse than the first transform in the sequence. Again, our battery of tests was restricted on points clouds data sets. Hence, the results showed in this paper may not be comparable with those of other papers. Baring that, those results may also be the consequence of an implementation error.

4 Experimentation

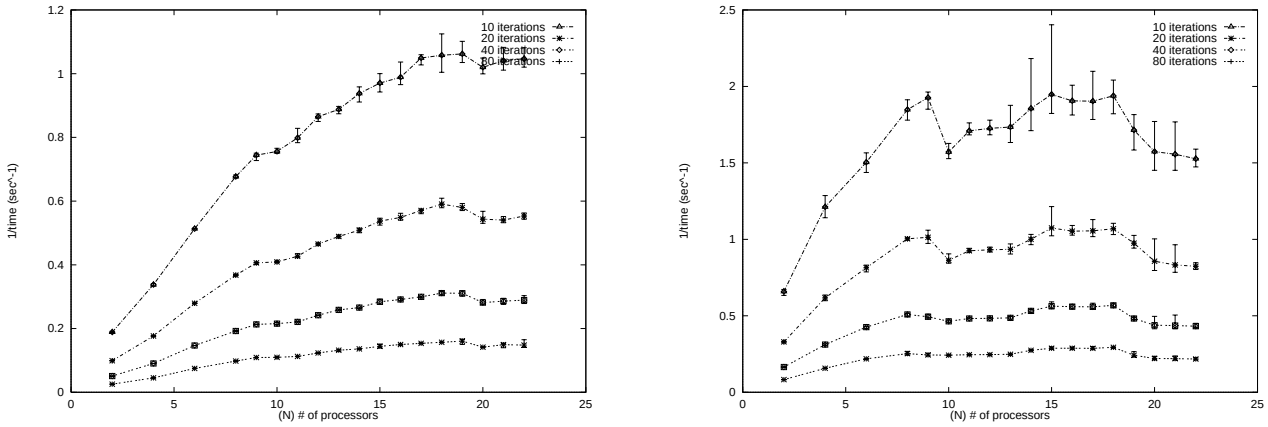
All experimentations were performed on a Beowulf class cluster of PCs comprising 24 Pentium II processors connected over a 100Mbps Ethernet switch. The nodes were configured as dual processors so that all CPUs fit within 12 boxes. The system was not completely symmetric: 9 of the PCs were dual Pentium II 350Mhz with 500MB of RAM, while the remaining 3 were dual Pentium II 450Mhz with 1GB of RAM. All processors were running the RedHat LINUX 5.2 operating system. We were initially inclined to use MPI as the message passing layer, but in preliminary tests, it was found that MPI did not support a true multicasting function. As multicasting was deemed to be important to our application, we decided to use PVM (version 3.3.11) which had an efficient implementation of multicasting and multityped messages.

Tests were run on eight image pairs acquired with a *Biris* range sensor [8]. For the sake of brevity, the results of only two of these test sets are reported here: a large pair ($n_r \approx 235K$, $n_f \approx 36K$), and a small pair ($n_r \approx 150K$, $n_f \approx 15K$). The results of the other image tests followed a pattern similar as those presented here, and led to the same conclusions. These images contained sufficient variation in shape so that they did not register trivially, and were considered to represent typical test cases.

Eighty iterations of the $pICP$ were executed for each image pair for a varying number of child processes, $N = 1, ..., 22$. Note that the total number of processors available in the system was 24, one of which was required for the parent process. The additional CPU which shared the same motherboard as the parent process CPU was not used, so that symmetry of the child processes was maintained.

A graph of the large and small image pair tests are respectively illustrated in Figures 8a) and b). The graphs show the system speed, defined as the inverse of the execution time as a function of N. The curves are the average of five different test runs with error bars indicating the minimum and maximum values between runs.

The large data plot (Fig. 8a)) shows that up to $N = 18$ the speed improvement is nearly linear in the number of child processes. For $N > 18$, the performance begins to degrade. The same effect is observed in the small data plot, but commencing at $N > 8$. In both cases, this degradation was believed to be caused by the increase in the communication overhead as N increased. The values of $N = 18$ and 8 therefore represent the limitations of the granularity of this implementation of the parallel ICP method on this system architecture for the respective data sets. It is believed that the observed limitations on the granularity will be similar for data sets of similar size, and this behavior



a) Large Image ($n_f \approx 36K$)

b) Small Image ($n_f \approx 15K$)

Figure 8: N vs. speed (inverse of time)

was indeed observed on the other data sets that were tested.

5 Conclusions

We have presented $pICP$, a parallel implementation of ICP which has achieved a linear speedup for up to 18 processors for an image of size $\approx 36K$ points. For a smaller image of size $\approx 15K$ points, the speedup was linear for up to 8 processors.

A number of enhancements were developed to improve the performance over that of the basic parallelization scheme. By distributing the cross-covariance calculation, it was possible to considerably reduce the communication overhead, which realized a speedup of $\approx 100\%$ in some runs. Shuffling the image served to better balance the load among the processes, which improved performance by another $\approx 30\%$. It was also found that a straightforward application of image thinning did not have any beneficial effects. While the iterations were certainly faster, the rate of reduction of the registration error did not improve. This result was deemed to be significant, as it was both counter-intuitive, and contrary to popular belief.

The amount of communication per process was quite small, and so the ultimate breakdown in linearity was attributed to specifics of the current networking system. It is believed that faster switches (which are currently available) or a different machine architecture (such as a shared memory machine) would reduce this effect so that the region of linearity could be extended over a much larger number of processes. Experiments on fine grain parallel systems are planned to assess the efficiency of the $pICP$ on such architectures.

References

[1] Paul J. Besl and Neil D. McKay. A method for registration of 3d shapes. *IEEE Trans. PAMI*, 14(2):239–256, February 1992.

[2] David A. Simon, Martial Hebert, and Takeo Kanade. Real-time 3-d pose estimation using a high-speed range sensor. In *IEEE Intl. Conf. Robotics and Automation*, pages 2235–2241, San Diego, California, May 8-13 1994.

[3] Piotr Jasiobedzki, Jimmy Talbot, and Mark Abraham. Fast 3d pose estimation for on-orbit robotics. In *ISR 2000: 31st Intl. Symposium on Robotics*, pages 434–440, Montréal, Canada, May 14-17 2000.

[4] Michael Greenspan and Guy Godin. A nearest neighbor method for efficient icp. *Submitted to the 3rd Int. Conf. on Three-Dimensional Imaging and Modeling*, 2001.

[5] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975.

[6] Ronald L. Rivest. On the optimality of elias's algorithm for performing best-match searches. In *Information Processing 74*, pages 678–681, 1974.

[7] B.K.P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal Optical Society of America A*, 4(4):629–642, April 1987.

[8] François Blais and Mario Lecavalier. Application of the Biris range sensor for volume evaluation. In *Optical 3-D Measurement Techniques III*, pages 404–413, Vienna, Austria, October 2-4 1995.

[9] T Masuda and N. Yokoya. A robust method for registration and segmentation of multiple range images. *Computer Vision and Image Understanding*, 61(3):293–307, 1995.