PARALLEL CLUSTERING AND DIMENSIONAL SCALING ON MULTICORE SYSTEMS

Xiaohong Qiu
Research Computing UITS
Indiana University Bloomington
Email: xqiu@indiana.edu

Geoffrey C. Fox, Huapeng Yuan, Seung-Hee Bae
Community Grids Laboratory
Indiana University Bloomington
gcf@indiana.edu yuanh@indiana.edu sebae@indiana.edu

George Chrysanthakopoulos, Henrik Frystyk Nielsen
Microsoft Research Redmond WA
georgioc@microsoft.com henrikn@microsoft.com

INVITED PAPER

KEYWORDS
Multicore, Grids, Data mining, Parallel Programming

ABSTRACT

Technology advances suggest that the data deluge, network bandwidth and computers performance will continue their exponential increase. Computers will exhibit 64-128 cores in some 5 years. Consequences include a growing importance of data mining and data analysis capabilities that need to perform well on both parallel and distributed Grid systems. We discuss a class of such algorithms important in Chemoinformatics, bioinformatics and demographic studies. We present a unified formalism and initial performance results for clustering and dimension reduction algorithm using annealing to avoid local minima. This uses a runtime CCR/DSS that combine the features of both MPI, parallel threaded and service paradigms.

1. INTRODUCTION

There are many important trends influencing scientific computing. One is the growing adoption of the eScience paradigm which emphasizes the growing importance of distributed resources and collaboration. Another is the data deluge with new instruments, sensors, and the Internet driving an exponential increase of data [1]. On the other hand, multicore chips are challenging because they require concurrency to exploit Moore's law in contrast to the improved architectures and increasing clock speed of the last 15 years that has allowed dramatic performance increase within a well established fixed (sequential) programming paradigm [2-4]. Thus we suggest that it is important to look at data analysis and data mining and derive efficient multicore implementations. The data deluge, its management in a distributed environment and its analysis (mining) are relevant for both eScience and commodity applications. The former could involve data from high throughput instruments used in Life Sciences. The latter includes the analysis of environmental and surveillance monitors or the data fetched from the Internet that could characteristic a user's interests. The RMS (Recognition, Mining, Synthesis) analysis from Intel [5, 6] identified data mining and gaming as critical applications for multicore chips. Scientific data is likely to be so voluminous that we need any implementation to work well on clusters of multicore chips with preferably the same programming model for the inter-chip as well as the intra-chip parallelism. On the other hand commodity applications might well not need cluster implementations but probably would choose thread-based runtimes involving managed code – Java or C#. Data is often distributed so the Grid capabilities are essential; data mining can be extremely computationally intense so parallel implementations will sometimes be necessary.

The importance of Grids and multicore to both eScience (scientific computing) and commodity applications, motivates us to look at scientific data mining but in a programming model that is natural for related commodity applications. This motivates the SALSA (Service Aggregated Linked Sequential Activities) [7] research that we describe here. SALSA is implementing a set of data mining applications on multicore systems using managed code (C#) with parallel synchronization from a runtime CCR (Concurrency and Computation Runtime) developed at Microsoft Research [12, 13]. CCR supports both MPI style synchronization and the dynamic threading essential in many concurrent commodity applications. Further there is a service model DSS (Decentralized System Services) built on top of CCR [14]. CCR is a possible choice of runtime that could bridge between scientific and commodity applications as it supports the key concurrent primitives used in both of them. SALSA proposes that one builds applications as a suite of services [8] rather than traditional subroutine or class libraries. The service model allows one to support integration within grid, cluster and inter-chip environments. Thus SALSA is exploring a possible future application (data mining) on multicore chips using a programming model that could be used across a broad set of computer configurations

Proceedings of the 2008 High Performance
Computing & Simulation Conference ©ECMS
Waleed W. Smari (Ed.)
ISBN: 978-0-9553018-7-2 / ISBN: 978-0-9553018-6-5 (CD)

and could be the basis of an approach that integrates scientific computing with commodity applications. We note that we program in a low level style with user responsible for explicit synchronization in the fashion that is familiar from MPI. There certainly could be general or domain specific higher level environments such as variants of automatic compilation, OpenMP, PGAS or even the new languages from Darpa's HPCS program [6, 15]. Our work can still be relevant here as it uses a runtime that is a natural target for such advanced high-level environments.

Performance is a critical question for any system that spans multiple different domains; integration of multiple paradigms requires that the performance is good in each paradigm. In previous papers [9-11], we have discussed the core performance of CCR and DSS and here we focus on applications and discuss in more detail their structure and performance.

The SALSA work is currently performed on a variety of two CPU multicore systems with a total of 4 or 8 cores and running variants of Linux and Windows operating systems. The results reported in this paper use a single 8 core machine termed Intel8b. This is a Dell Precision PWS690, with 2 Intel Xeon CPUs x5355 at 2.66GHz, an L2 Cache 2X4M, 4GB Memory, and running Vista Ultimate 64bit.

In the following section we briefly discuss our programming model and refer the reader to other papers [9-11] for more details. In section 3, we discuss the data mining algorithms investigated and give some overall performance results. In section 4, we summarize our results and identify the key features of the application structure and the implications for the parallel run time. Conclusions are in section 5.

2. PARALLEL CCR RUNTIME

CCR provides a framework for building general collective communication where threads can write to a general set of ports and read one or more messages from one or more ports. The framework manages both ports and threads with optimized dispatchers that can efficiently iterate over multiple threads. All primitives result in a task construct being posted on one or more queues, associated with a dispatcher. The dispatcher uses OS threads to load balance tasks. The current applications and provided primitives support a dynamic threading model with capabilities that include:

1) *FromHandler*: Spawn threads without reading ports
2) *Receive*: Each handler reads one item from a single port
3) *MultipleItemReceive*: Each handler reads a prescribed number of items of a given type from a given port. Note items in a port can be general structures but all must have same type.
4) *MultiplePortReceive*: Each handler reads a one item of a given type from multiple ports.
5) *JoinedReceiv*e: Each handler reads one item from each of two ports. The items can be of different type.
6) *Choice*: Execute a choice of two or more port-handler pairings
7) *Interleave*: Consists of a set of arbiters (port -- handler pairs) of 3 types that are Concurrent, Exclusive or Teardown (called at end for clean up). Concurrent arbiters are run concurrently but exclusive handlers are not.

One can spawn handlers that consume messages as is natural in a dynamic search application where handlers correspond to links in a tree. However one can also have long running handlers where messages are sent and consumed at a rendezvous points (yield points in CCR) as used in traditional MPI applications. Note that "active messages" correspond to the spawning model of CCR and can be straightforwardly supported. Further CCR takes care of all the needed queuing and asynchronous operations that avoid race conditions in complex messaging. CCR is attractive as it supports such a wide variety of messaging from dynamic threading, services (via DSS described in [9]) and MPI style collective operations.

```
Main Routine for Exchange Pseudocode {
    Create CCR dispatchers to control threads
    Create a queue to hold tasks
    Set up start ports with MPI initialization data such as thread
      number
    Invoke handlers (MPI threads) on start ports
} End Main Routine

MPI logical thread Pseudocode (Arguments are start port
contents) {
    Calculate nearest neighbors for exchange collective
    Loop over stages  { Post information to 2 ports that will be
      read by left and right neighbors

      yield return on CCR MultipleItemReceive will wait till this
      thread's information is available in its ports and continue
      execution after reading 2 ports

      Do computation for this stage
    } End loop over stages

    Each thread sends information to ending port
    and thread 0 only does yield return on CCR
    MultipleItemReceive to collect information from all threads to
    complete run after reading from one port for each thread
    (this is a reduction operation).
} End MPI Thread
```

CODE SAMPLE 1: MPI EXCHANGE IN CCR

We have [11] already compared CCR with MPI and note that posting to a port in CCR corresponds to a MPISEND and the matching MPIRECV is achieved from arguments of handler invoked to process the port. MPI has a much richer set than CCR of defined methods that describe different synchronicity options, various utilities and collectives. These include the multi-cast (broadcast, gather-scatter) of messages with the calculation of associative and commutative

functions on the fly. It is not clear what primitives and indeed what implementation will be most effective on multicore systems and so we have not performed an exhaustive study of MPI collective patterns in CCR. In fact it is possible that SALSA's results which suggest one can support in the same framework a set of execution models that is broader than today's MPI, could motivate a new look at messaging standards for parallel computing. CCR only has built-in primitives to support MPI shift and reduction operations but we exploited CCR's ability to construct customized collectives sketched in Code Sample I to implement the MPI Exchange pattern [11]. An important innovation of the CCR is to allow sequential, asynchronous computation without forcing the programmer to write callbacks, or continuations, and at the same time not blocking an OS thread. This allows the CCR to scale to tens of millions of pending I/O operations, but with code that reads like synchronous, blocking operations.

Note that all our work was for managed code in C# which is an important implementation language for commodity desktop applications although slower than C++. In this regard we note that there are plans for a C++ version of CCR which would be faster but prone to traditional un-managed code errors such as memory leaks, buffer overruns and memory corruption. The C++ version could be faster than the current CCR but eventually we expect that the C# CCR will be within 20% of the performance of the C++ version. CCR has been extensively applied to the dynamic threading characteristic of today's desktop application but its largest use is in the Robotics community. One interesting use is to add an efficient port-based implementation of "futures" to C#, since the CCR can easily express them with no modifications in the core runtime. CCR is very portable and runs on both CE (small devices) and desktop windows. DSS sits on top of CCR and provides a lightweight, service oriented application model that is particularly suited for creating Web/Grid-style applications as compositions of services running in a distributed environment. Its use in SALSA with performance results is described in [9, 10] and very briefly in Section 4 of this paper.

3. DATA MINING

In this paper we consider data mining algorithms that analyze a set of N data points $\underline{X}(x)$ labeled by x in a D dimensional space. These algorithms have a common formalism corresponding to iterative minimization of the function F given by equations (1) and (2).

$$F = -T\sum_{x=1}^{N} a(x)\ln Z(x) \text{ where} \quad (1)$$

$$Z(x) = \sum_{k=1}^{K} g(k)\exp[-0.5(\underline{X}(x) - \underline{Y}(k))^2 / (Ts(k))] \quad (2)$$

There are four useful algorithms covered by the above: Clustering with Deterministic Annealing (CDA) [16-19]; Gaussian Mixture Models (GMM) [21]; Gaussian Mixture Models with DA (GMMDA) [22]; Generative Topographic Maps (GTM) [20]. We show how equations (1) and (2) cover each of these cases below. Note that for GMM and GTM, F is directly the cost function C (or negative of log likelihood) while for the annealing cases CDA and GMMDA, F is C-TS, the "free energy" where T is a temperature and S is the Shannon Entropy [18]. The sum over k corresponds to sum over clusters or mixture model components. A key characteristic of all these algorithms is "missing data" represented by the sum over clusters k in equation (2). We do not know a priori which value of k (e.g. which cluster) is associated with each data point $\underline{X}(x)$. This missing data characteristic also allows the applicability of the well known EM method [21] which is similar to steepest descent and can be shown to always decrease the objective function F in all four cases. Steepest descent methods are prone to find local minima so DA is attractive as it mitigates the effect of local minima.

In the annealing method one includes the entropy associated with these degree of freedom k and minimizes the Free Energy. The temperature is varied in an annealing schedule from high values (when F is dominated by entropy) to low values when the true cost C dominates. Unlike simulated annealing, DA involves no Monte Carlo but rather optimizes (1, 2) iteratively as temperature T is varied from high to low values. For clustering CDA improves on the well known K-means clustering algorithm [20]. In our cases the annealing can be interpreted as a multi-scale approach with $T^{1/D}$ as a distance scale. Now we define the four methods for which equations (1, 2) can be used and after that discuss their solution and our approach to parallelism.

For the first example, CDA clustering, the variables in (1) are given by:

$$a(x) = 1/N,\ g(k)=1,\ s(k) = 0.5 \quad (3)$$

and T is temperature decreased to 1 by some schedule. DA finds K cluster centers $\underline{Y}(k)$ where K is initially 1 and is incremented by algorithm as T decreases.

We emphasize that unlike K-Means [19] or GMM, one need not specify the number of clusters K a priori in CDA. Rather K is determined by the annealing; as the distance scale decreases (see example in fig. 5 later), more clusters are determined. In the extreme limit T=0, all points x become clusters of size one and K=N.

For the second example, Gaussian Mixture Models GMM are defined by:

$$a(x)=1,\ g(k)= P_k/(2\pi\sigma(k)^2)^{D/2},\ s(k)= \sigma(k)^2 \quad (4)$$

The component probability P_k, the standard deviation $\sigma(k)$ and component center Y(k) are varied with number of components K fixed a priori.

Equation (3) specializes to a common case of spherical distributions. The general case has $s(k)$ as a general symmetric DxD correlation matrix but this does not impact key ideas; it just makes the formalism more complex. Of course the model components in GMM are

"just" clusters but GMM is more natural than clustering when the components have very different sizes. Although GMM makes P_k and $\sigma(k)$ as fitted variables, the formulae for their estimated values (using EM Method) are in fact identical to those from clustering. So CDA can find variable sized clusters although GMM could be a more precise approach.

One can easily extend GMM to add annealing [22] although there is currently little practical experience. This leads to GMMDA, which is given by:

$$a(x)=1,\ g(k)= \{P_k/(2\pi\sigma(k)^2)^{D/2}\}^{1/T},\ s(k)= \sigma(k)^2 \qquad (5)$$

and T is temperature decreased to 1 by some schedule. GMMDA finds K component probabilities P_k, standard deviations $\sigma(k)$ and component centers Y(k) where K is initially 1 and is incremented by algorithm as T decreases.

The final algorithm considered here has a very different goal to GMM and DAC; namely it addresses dimensional scaling or the derivation of a set of vectors $\underline{v}_i$ in a metric space where the distance between vectors *i* and *j* is given by a known discrepancy function δ_{ij}. Here δ_{ij} may come from the distance between points *i* and *j* in another vector space or be a discrepancy derived from an algorithm like BLAST comparing sequences in bioinformatics. In particular, we look at a powerful algorithm GTM (Generative Topographic Mapping) developed in [20] and often used to map from high dimensional spaces to two or three dimensions for visualization. This is illustrated in fig. 1 showing the 2D GTM projection of a set of three Gaussians in a 5D space. Note that one could use the simple Principal Component Analysis (PCA) approach [23]. This gives an optimal linear projection but does not perform well on complex problems whereas GTM has a nonlinear algorithm that is broadly effective [20]. PCA gave poor results on many Cheminformatics problems in high dimensions as the top two eigenvectors (used by PCA for 2D projection) do not capture much of the structure. Fig. 2 shows a successful GTM projection of two clusters in a D=155 dimensional Cheminformatics case. GTM is defined in the syntax of Equation (1) by:

$a(x) = 1$; $g(k) = (1/K)(\beta/2\pi)^{D/2}$; $s(k) = 1/\beta$; $T = 1$ and β and $\underline{W}_m$ are varied for fixed K, $\underline{L}(k)$, and M below. $\underline{L}(k)$, $\underline{\lambda}$ and $\underline{\mu}_m$ are vectors in the latent (2D) space.

$$\underline{Y}(k) = \sum_{m=1}^{M} \underline{W}_m \phi_m(\underline{L}(k)) \text{ with fixed} \qquad (6)$$

$$\phi_m(\underline{\lambda}) = \exp(-0.5(\underline{\lambda} - \underline{\mu}_m)^2 / \sigma^2)$$

GTM has excellent scaling properties and works well on large problems. We are currently applying it to the over 10 million compounds in PubChem. GTM can used for both clustering and dimensional reduction but we use DA for clustering and GTM just for the projection to 2D. The clusters found in GTM are viewed as a convenient averaging of the high dimensional space. For example in figs. 1 and 2 with 3 or 2 "real" clusters respectively, our GTM used *K*=225 averaging clusters. This large number of clusters used in averaging for projection leads to exceptional parallel performance for GTM given below. As discussed above, GTM and DA clustering are essentially the same algorithm with different optimizations; one for mapping and the other for robust clustering. Note GTM is closely related to SOM (Self Organizing Maps) but there are other important dimensional scaling methods whose parallel implementation we are also working on. The classic MDS (Multi Dimensional Scaling) approach using the SMACOF algorithm [23] has the advantage that it preserves distances δ_{ij} and not just the spatial topology mapped by GTM. Further it does not need vectors in the original space but just their discrepancies δ_{ij}. Other interesting methods with this property include random projection [24] and quadratic programming methods [25]. We will report on the multicore implementation of these other projection methods later. Here we just look at GTM which is a powerful efficient tool when the original vectors $\underline{X}(x)$ are known. We note that one could add annealing to GTM but we have not explored this yet.

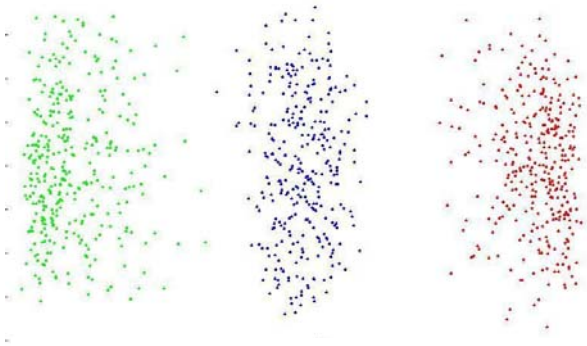


Figure 1: GTM Projection of a simple test problem with three Gaussians in a D=5 dimensional space

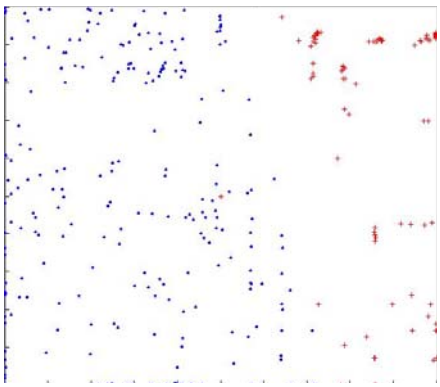


Figure 2. GTM Projection for 2 clusters from DA in space of 155 Chemical Properties labeled as a . or +

For the four algorithms defined in equations (3-6), solution of (1, 2) is implemented by a variation of the Expectation Maximization (EM) algorithm [21]:

$$\underline{Y}(k) = \sum_{x=1}^{N} \underline{X}(x) \Pr[\underline{X}(x) \in C(k)] / \sum_{x=1}^{N} \Pr[\underline{X}(x) \in C(k)] \quad (7)$$

$$\Pr[\underline{X}(x) \in C(k)] = \exp[-0.5(\underline{X}(x) - \underline{Y}(k))^2 / T] / Z(x) \quad (8)$$

written for the case of DA clustering where new values of cluster centers $\underline{Y}(k)$ are calculated iteratively from probabilities of x belonging to cluster $C(k)$. GTM, GMM and GMMDA have similar formulae with more quantities being calculatedly but always as averages with probabilities that a point $\underline{X}(x)$ belongs to a component/cluster k.

Realistic implementations must support both conventional real valued quantities in the above equations and also binary variables (for chemical fingerprints) and profiles in bioinformatics where the variables represent the frequencies with which features occur.

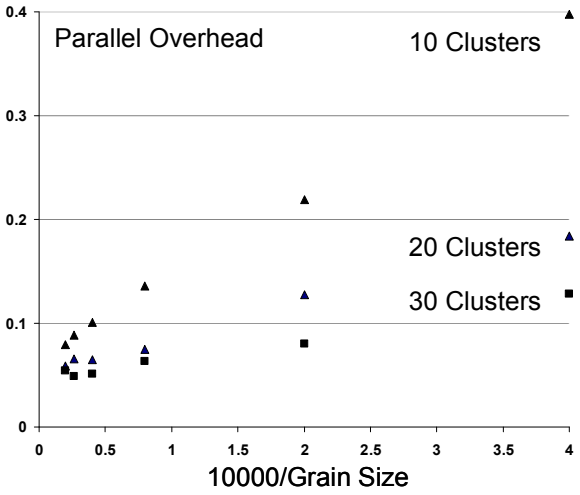


Figure 3. 8 core Parallel Overhead (approximately 1-efficiency) from Equation (9) for GIS 2D DA Clustering on Intel8b for three values (10, 20,30) of the number of clusters and plotted against 10000/N

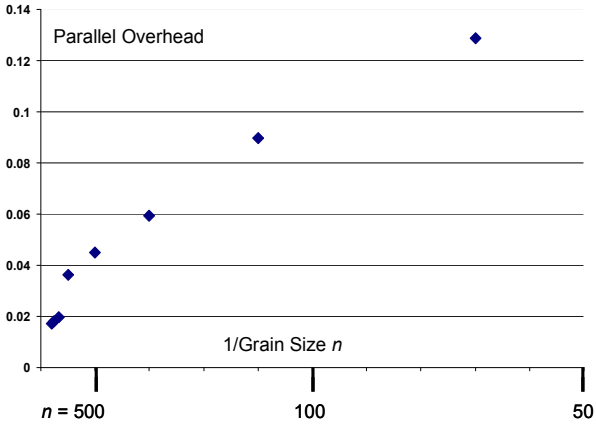


Figure 4. 8 core Parallel Overhead defined in Eq. (9) on Intel8b plotted against 8/N for GTM using M=256 and K=4096

Initial results on the parallel performance of DA clustering are shown in fig. 3 for runs on the 8 core Intel machine Intel8b used in all results presented in this paper. The figure shows that DA has a parallel overhead [26, 27] that decreases asymptotically like 1/grain size as the data set increases. Here grain size n is the dataset size N divided by the number of processors (cores) which is here 8. Putting $T(P)$ as the execution time on P cores, we can define:

$$\text{Overhead } f = (PT(P) - T(1))/T(1) \quad (9)$$

$$\text{Efficiency } \varepsilon = 1/(1+f) = \text{Speed up } S(P)/P \quad (10)$$

Thus the overhead of 0.05 seen for large n (small $1/n$) in fig. 3 corresponds to an excellent speedup of 7.6. The results for GTM in fig. 4 show even smaller overheads even at small grain size due to the substantial additional computation (matrix multiplication and equation solving) in this case. We emphasize that much of the critical overhead in multicore parallel code is not synchronization but rather due to interference between cores in the memory subsystem.

Tables 1, 2 and 3 study the parallel overhead for GTM as a function of the variables N (number of points), K (number of averaging clusters), M (number of mapping functions). The overhead lies between .01 (speedup of 7.9) and .05 (speedup of 7.6) except for the "small problem" N=1000 data points in table 1, where the overhead rises to 18% for the smallest problem. These results emphasize the excellent parallel efficiency of these algorithms and that large problems run well!

Table 1: Parallel Overhead for GTM as function of M

M=	128	256	512	768	1024	1280
K=16384 N=20000	-	0.014	0.013	0.014	0.016	0.022
K=900 N=1000	0.18	0.17	0.16	0.09	-	-

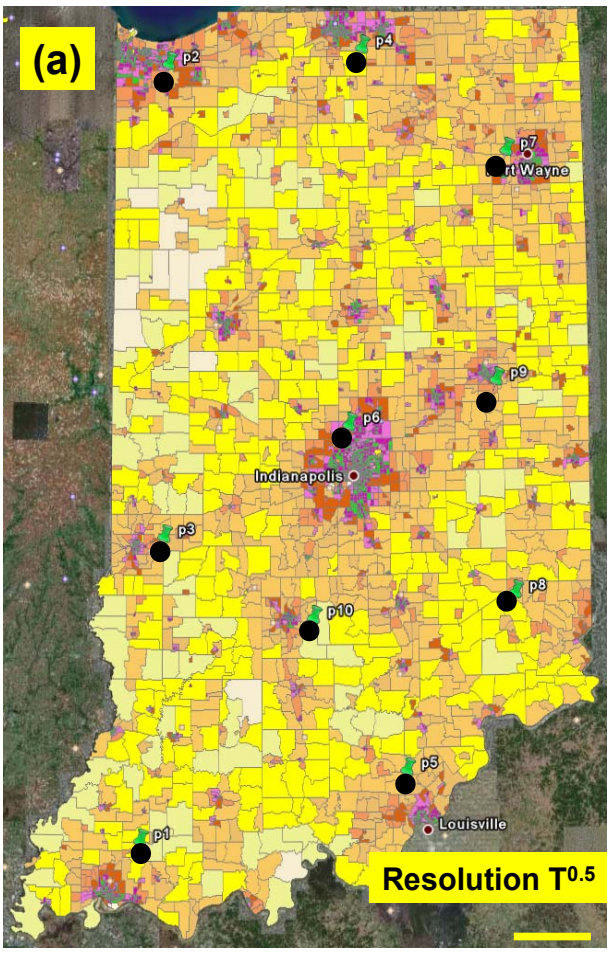
Table 2: Parallel Overhead for GTM as function of K

K=	1024	2304	4096	6400	9216	12544
N=20000 M=256	0.026	0.023	0.018	0.017	0.028	0.015

Table 3: Parallel Overhead for GTM as function of N

N=	4000	8000	12000	16000	20000
K=4096 M=256	0.045	0.036	0.020	0.018	0.017

In fig. 5, we illustrate the multi scale aspect of DA clustering with results from the clustering of the State of Indiana Census data for two temperatures in the annealing schedule. The value of the temperature is represented by a distance $T^{0.5}$ on figures 5(a) and 5(b). At the higher temperature in fig. 5(a), one finds 10 clusters. The larger cities in Indiana (such as the capital Indianapolis) are identified but some other municipalities are averaged together and not found individually. As we lower resolution in fig. 5(b), there are 30 clusters and most of the major towns in Indiana are identified. This figure shows possible studies one can perform as it looks at total population as well as three different groupings: Hispanics, Asians and renters with somewhat different clustering results.



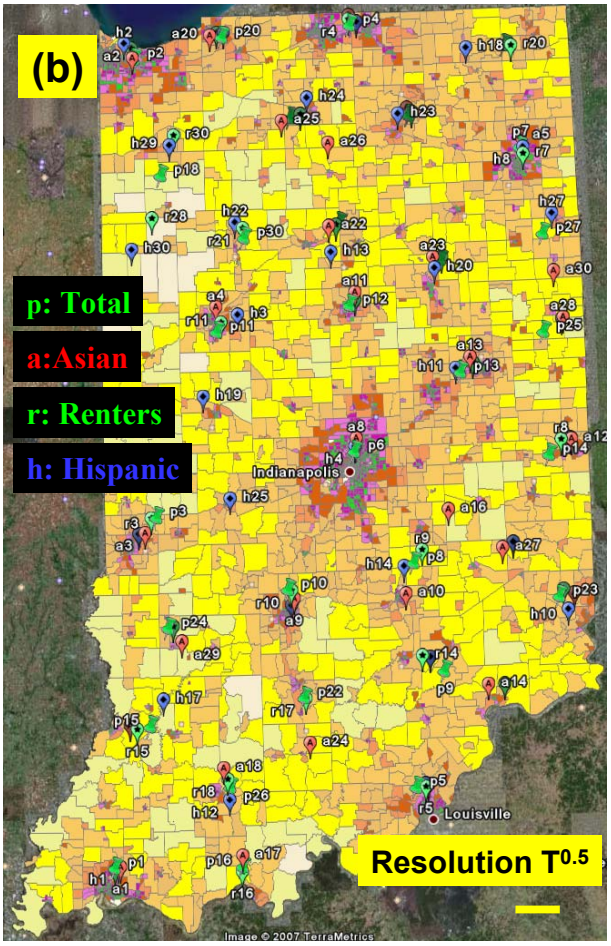


Figure 5. DA Clustering of Census data for state of Indiana showing K=10 clusters in (a) and 30 in (b). The resolutions $T^{0.5}$ are shown in bottom right. (a) shows just total population while (b) also shows Asian, Hispanic and Renter communities

4. PARALLEL PROGRAMMING

The algorithms illustrated in equations (1-8) have a structure familiar from many scientific computing areas [6, 26, 27]. There is an iteration – in this case over the annealing schedule for T and the steps needed for the EM method to converge. Synchronization is needed at the end of each iteration. Further looking into more detail, we find that the iteration consists of sums like Equations (7) and (8) calculating vector and matrix elements combined with linear algebra. The latter is identification of principal directions for CDA and GMMDA. There is no significant linear algebra for GMM while GTM needs matrix multiplication and linear equation solution. The sums themselves are first calculated in the memory of the threads and then after synchronization, accumulated into "global" variables.

This strategy assures good use of cache with negligible interference that occurs when two cores write to different memory locations that share the L1 cache [11]. Thus we see that all these algorithms have a "loosely synchronous" structure where the parallel algorithm consists of "compute-synchronize" stages where synchronization typically implies all cores reach a barrier [6, 27]. CCR supports the loosely synchronous paradigm with modest overheads analyzed in detail in earlier papers. Although CCR supports messaging like MPI, we only need CCR for synchronization in the applications considered in section 3. Data communication is achieved by accessing it in the memory shared by the threads. This is attractive as read access to shared information does not incur cache interference penalties. One does not need to put in thread memory the "edges" of regions as in Halos or ghost cells familiar in MPI. Rather one needs cache-sized blocks of data copied into thread memory; that is performed by the thread itself and not by communicating between threads. The critical source of overhead on a multicore chip is the memory subsystem.

Comparing our multicore implementations with traditional parallel programming, we see that we are using essentially the same programming model with domain decomposition breaking up the application into parallel components that execute in a loosely synchronous fashion. We use threads not processes in each core which allows us to optimize data connection between the decomposed components. Note we do need to link our thread based model inside a multicore system with a traditional distributed memory model if our algorithm needs parallelism across a cluster. The model

for parallelism is identical inside and outside the multicore system but the data connection is different.
The fine grain parallelism is handled by CCR but this is not a complete software engineering model as it does not provide the desired modularity. Here we are using services as the building block. Services are attractive as they allow linkage to the distributed (Grid) programming model. We have successfully used DSS in in our early work. This is a Grid compatible service model which runs with high performance inside a chip. Further DSS is built on top of CCR which we use for synchronization inside the multicore and will use for linking to MPI for cluster operations. DSS has latencies of around 35 μs which corresponds to between 0.25 and 0.5 (floating point) million operations on an 8 core system achieving 1-2 Gflops per core. This implies that for example linear algebra on 100x100 matrices can be packaged as services without significant overhead. We have used DSS to encapsulate data reading, manipulation and visualization and will extend to break up the data mining itself in later work.

5. CONCLUSIONS

SALSA aims to develop scalable parallel data mining algorithms with good multicore and cluster performance and understand needed software runtime and parallelization method. We use managed code (C#) and package algorithms as services to encourage broad use assuming experts parallelize core algorithms. We have shown that Microsoft CCR supports well MPI, dynamic threading and via DSS a service model of computing. Detailed performance measurements give speedups of 7.5 or above on 8-core systems for "large problems" with deterministic annealed algorithms for clustering, Gaussian Mixtures, GTM and MDS (dimensional reduction). Future work includes further algorithms and applications as well as extensions to distributed and cluster implementations.

REFERENCES

[1] Tony Hey and Anne Trefethen, *The data deluge: an e-Science perspective* in "Grid Computing: Making the Global Infrastructure a Reality" edited by Fran Berman, Geoffrey Fox and Tony Hey, John Wiley & Sons, Chicester, England, ISBN 0-470-85319-0, February 2003

[2] Jack Dongarra Editor *The Promise and Perils of the Coming Multicore Revolution and Its Impact*, CTWatch Quarterly Vol 3 No. 1 February 07, http://www.ctwatch.org/quarterly/archives/february-2007

[3] David Patterson *The Landscape of Parallel Computing Research: A View from Berkeley 2.0* Presentation at Manycore Computing 2007 Seattle June 20 2007 http://science.officeisp.net/ManycoreComputingWorkshop07/Presentations/David%20Patterson.pdf

[4] Annotated list of multicore Internet sites http://www.connotea.org/user/crmc/

[5] Pradeep Dubey *Teraflops for the Masses: Killer Apps of Tomorrow* Workshop on Edge Computing Using New Commodity Architectures, UNC 23 May 2006 http://gamma.cs.unc.edu/EDGE/SLIDES/dubey.pdf

[6] Geoffrey Fox tutorial at Microsoft Research *Parallel Computing 2007: Lessons for a Multicore Future from the Past* February 26 to March 1 2007.

[7] Home Page for SALSA Project at Indiana University http://www.infomall.org/salsa and http://www.infomall.org/ has links to publications and presentations of SALSA group.

[8] Dennis Gannon and Geoffrey Fox, *Workflow in Grid Systems* Concurrency and Computation: Practice & Experience 18 (10), 1009-19 (Aug 2006), Editorial of special issue prepared from GGF10 Berlin

[9] Xiaohong Qiu, Geoffrey Fox, and Alex Ho Analysis of Concurrency and Coordination Runtime CCR and DSS, Technical Report January 21 2007

[10] Xiaohong Qiu, Geoffrey Fox, H. Yuan, Seung-Hee Bae, George Chrysanthakopoulos, Henrik Frystyk Nielsen *High Performance Multi-Paradigm Messaging Runtime Integrating Grids and Multicore Systems*, published in proceedings of eScience 2007 Conference Bangalore India December 10-13 2007

[11] Xiaohong Qiu, Geoffrey C. Fox, Huapeng Yuan, Seung-Hee Bae, George Chrysanthakopoulos, Henrik Frystyk Nielsen *Performance of Multicore Systems on Parallel Data Clustering with Deterministic Annealing* Proceedings of ICCS Krakow Poland June 23-25 2008.

[12] Microsoft Robotics Studio is a Windows-based environment that includes end-to-end Robotics Development Platform, lightweight service-oriented runtime, and a scalable and extensible platform. For details, see http://msdn.microsoft.com/robotics/

[13] Georgio Chrysanthakopoulos and Satnam Singh "An Asynchronous Messaging Library for C#", Synchronization and Concurrency in Object-Oriented Languages (SCOOL) at OOPSLA October 2005 Workshop, San Diego, CA. http://urresearch.rochester.edu/handle/1802/2105

[14] Henrik Frystyk Nielsen, George Chrysanthakopoulos, "Decentralized Software Services Protocol – DSSP" http://msdn.microsoft.com/robotics/media/DSSP.pdf

[15] Internet Resource for HPCS Languages http://crd.lbl.gov/~parry/hpcs_resources.html

[16] Geoff M. Downs, John M. Barnard *Clustering Methods and Their Uses in Computational Chemistry*, Reviews in Computational Chemistry, Volume 18, 1-40 2003

[17] Kenneth Rose, Eitan Gurewitz, and Geoffrey C. Fox *Statistical mechanics and phase transitions in clustering* Phys. Rev. Lett. 65, 945 - 948 (1990) http://dx.doi.org/10.1103/PhysRevLett.65.945

[18] Rose, K. *Deterministic annealing for clustering, compression, classification, regression, and related optimization problems*, Proceedings of the IEEE Vol. 86, pages 2210-2239, Nov 1998

[19] *K-means algorithm* at Wikipedia http://en.wikipedia.org/wiki/K-means_algorithm

[20] Bishop, C. M., Svensen, M., Williams, C. K. I. *GTM: The generative topographic mapping*. Neural Comput. 1998, 10, 215-234.

[21] Dempster, A.P., Laird, N.M., & Rubin, D.B. (1977). *Maximum-likelihood from incomplete data via the EM algorithm*. J. R. Statist. Soc. Ser. B (methodological), 39, 1–38.

[22] Naonori Ueda and Ryohei Nakano *Deterministic annealing EM algorithm* Neural Networks Volume 11, Issue 2, 31 March 1998, Pages 271-282 http://dx.doi.org/10.1016/S0893-6080(97)00133-0

[23] Ingwer Borg, Patrick J. F. Groenen *Modern*

Multidimensional Scaling: Theory and Applications Springer August 2005 ISBN-10: 0387251502

[24] Golan Yona *Methods for Global Organization of the Protein Sequence Space* PhD Thesis Hebrew University 1999 http://www.cs.cornell.edu/golan/Thesis/thesis.ps.gz

[25] E. Halperin, J. Buhler, R. Karp, R. Krauthgamer and B. Westover *Detecting protein sequence conservation via metric embeddings* Bioinformatics Vol. 19 Suppl. 1 2003 Pages i122-i129

[26] "The Sourcebook of Parallel Computing" edited by Jack Dongarra, Ian Foster, Geoffrey Fox, William Gropp, Ken Kennedy, Linda Torczon, and Andy White, Morgan Kaufmann, November 2002.

[27] Fox, G. C., Messina, P., Williams, R., "Parallel Computing Works!", Morgan Kaufmann, San Mateo Ca, 1994.