MPI-ClustDB: A fast String Matching Strategy utilizing Parallel Computing

Thomas Hamborg, Jürgen Kleffe

Institut für Molekularbiologie und Bioinformatik
Charité-Universitätsmedizin Berlin
Arnimallee 22
{thomas.hamborg,juergen.kleffe}@charite.de

Abstract: ClustDB is a tool for the identification of perfect matches in large sets of sequences. It is faster and can handle at least 8 times more data than VMATCH, the most memory efficient exact program currently available. Still ClustDB needs about four hours to compare all Human ESTs. We therefore present a distributed and parallel implementation of ClustDB to reduce the execution time. It uses a message-passing library called MPI and runs on distributed workstation clusters with significant runtime savings. MPI-ClustDB is written in ANSI C and freely available on request from the authors.

1 Introduction

Since many bioinformatics problems deal with the analysis of large amounts of data, parallel computing has proven to be an important tool to ensure computation capability and computation in reasonable time. In addition to traditional massively parallel computers, distributed workstation clusters play an increasing role in scientific computing. But so far, there had been little success in using distributed computing for large scale sequence matching. MUMMER [Ku04] and VMATCH [AKO04] are the most sophisticated programs implementing suffix tree and suffix array algorithms for simultaneous sequence matching. But due to their high memory usage these algorithms are not able to deal with datasets as large as necessary. Moreover both algorithms are not parallelizable for distributed memory architectures. Futamura et al. [FAK98] suggested an algorithm for parallel sorting of suffixes which performs a bucket sort of suffixes followed by parallel sorting of buckets. Still the algorithm requires large RAM by storing all sequences and suffix positions simultaneously. Another drawback is that the sorting of buckets cannot be performed using optimal algorithms. Hence, depending on the data, the method does not always improve overall computing time. Two other attempts of using massively parallel computation are the approaches by Iliopoulos et al. [IK02] and Kalyanaraman et al. [Ka03] which use far more memory than is available for practical input sizes. The latter publication estimates the demand of 512 parallel processors each with 512 MB RAM in order to compare five million human ESTs. In contrast ClustDB [KMW06] uses a new sorting algorithm named

partitioned suffix array method. It permits working with at least 8 times more data than VMATCH and MUMMER and offers several ways of efficient parallel computing. We therefore developed the program MPI-ClustDB that significantly reduces time consumption for a group of loosely coupled computers. This parallel approach allows to compare about six million human ESTs using only 7 personal computers each equipped with a 2.6 GHz Intel Pentium 4 CPU and 2 GB RAM. MPI-ClustDB is designed as a data-parallel approach where in certain parts of the program one has to cope with a variable number of identical tasks and each process executes more or less the same set of commands on its data (task). MPI, the de facto standard for distributed memory systems, is used for inter-process communication. It supports dynamic assignment of tasks to processes and has the advantage of running on several platforms without code alteration.

2 Algorithm

MPI-ClustDB is designed in a Master/Slave-manner where one process coordinates the scheduling and allocates tasks to a number of slave processes. It is assumed that all processes have access to the entire data. Therefore the master converts the input data into a fast accessable and space saving format called DNA_Stat database and distributes suchlike data across the slaves. The aim is the identification of all matching substrings of a certain minimal length in a large set of sequences which are derived by performing two computational steps called *Start Word Sort* and *Substring Detection*. We describe these steps together with the associated parallelization strategies in section 2.1 and 2.2. Subsequently the results are converted into a user-friedly output format. The parallelization of the conversion is described in section 2.3.

2.1 Start Word Sort

Let L be the number of nucleotides of the concatenated sequence formed from all considered sequences separated by a dot character. Conventional suffix array methods store and sort a vector of pointers of length L into the concatenated sequence. These pointers are called suffix positions and require at least four times more memory than needed to store the sequence. We therefore cut the vector of suffix positions into N pieces of length L/N which are processed one by one. A word length W between 3 and 10 is fixed and the pointers in each subvector are sorted in lexicographic order of the first W characters the suffixes begin with (Fig.1 - Step 1). Then each of the N sorted subvectors splits into blocks of suffix positions which start with the same word and are equally colored in Fig. 1. The parameter N is determined by the RAM size as each subvector has to fit into RAM for efficient sorting. In the parallel approach we choose N at least as large as the number of slave processes. The master process sends the appropriate sequence regions (two numbers) to the slaves and receives the sorted suffix positions. Each slave processor works in linear time $O(L/N)$ and requires $L/3 + 4(L/N + Z)$ bytes of RAM in order to store the

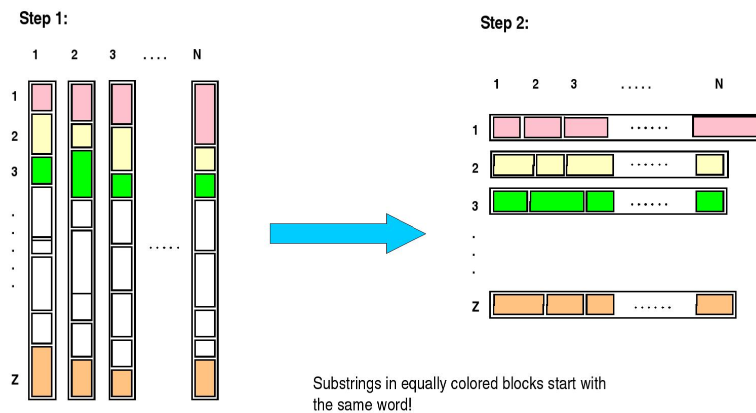

Figure 1: partitioned suffix array algorithm

complete sequence in compressed form (1 byte for 3 nucleotides), L/N suffix positions and $Z = 4^W$ different word counts. Let p be the number of available processors, then the total runtime for this step is roughly $O(L/p)$.

2.2 Substring Detection

In step 2 all blocks of suffix positions starting with the same word are collected from the N subvectors formed in step 1 and merged into Z new vectors displayed horizontally in Fig. 1 - Step 2. Z is the number of different words of length W. Each of these vectors is individually scanned for repeats of length M. Details about the calculations are given in [KMW06]. In case of parallel computing the master sends the Z vectors to the slaves and the slaves calculate the positions of multiple substrings which form clusters. Contrary to parallelization in step 1, dynamic allocation of tasks is indispensable here. Some words occur more frequently than others and hence the Z vectors differ greatly in length. But even if two start words occur with equal frequencies, the clusters originating from them will usually differ in size and so will the corresponding times of computation.
This step is performed in $O(M * L/(p * W))$ time requiring $L/3 + 8 * F$ bytes of RAM for each slave process where F is the maximum frequency of all considered start words. Assuming we have L bytes of RAM, we can use approximately $2 * L/3$ bytes for a table of size $8 * F$ that is necessary for the iterated suffix sort algorithm described in [KMW06], i.e. F can be as large as $L/12$. About every twelfth of all overlapping start words must be the same in order to cause failure of the algorithm with L bytes of RAM. In general F rapidly decreases by increasing word length.

2.3 Output Conversion

Subsequent to the partitioned suffix array algorithm each set of multiple substrings is represented by a cluster number and a set of global sequence positions in the concatenated sequence. These positions have to be turned into sequence numbers according to the succession in the input file and local sequence positions in the respective sequences. This task is carried out by means of a binary search algorithm. We use a scheduling strategy called *fixed-size chunking* [Ha97] here. A fixed amount of positions from substring cluster elements is sent to a slave, the sequence numbers and local positions are calculated and sent back to the master. The computation time for this part is $O(L * logS/p)$ where S denotes the number of sequences. Each processor requires $4 * S$ bytes of RAM in order to store the start positions of all individual sequences.

3 Implementation

MPI-ClustDB processes DNA-sequence data in the established formats Genbank, EMBL and FASTA or in DNA_Stat database format. The latter is an inhouse binary format that allows for fast direct access of individual sequences and playes a keyrole in fast data communication. It significantly reduces runtime especially if MPI-ClustDB repeatedly runs on the same data. Results of the substring calculation are presented in a tabular form with the three columns cluster number, sequence number and match position. It is possible to obtain the results in text file and/or DNA_Stat database format. Summary results are written to a seperate log file and several options of the program are described in [KMW06].
In order to execute MPI-ClustDB, an implementation of the Message Passing Interface communication protocol has to be installed. A large number of implementations is freely available. We have choosen the widely spread MPICH2 implementation that can be obtained from http://www-unix.mcs.anl.gov/mpi/mpich2/. As we make use of standard MPI commands only, it should be possible to link against any other MPI library, too. However, it is important to use buffered and blocking MPI send/receive functions in order to avoid deadlocks.

4 Results

We investigate the speedup of MPI-ClustDB compared to the serial ClustDB implementation for a 100 MBit/s and 1000 MBit/s ethernet network connection. If T_0 denotes the runtime of the serial solution and T_p denotes the runtime of the parallel solution with p processes, speedup is defined as $S_p = T_0/T_p$. All computations were performed on a test cluster consisting of seven standard personal computers. Each of them has a 2.6 GHz Intel Pentium 4 CPU and 2 GB of RAM running the operating system Mandriva Linux 2006.
We report application to the set of all 6,054,053 human ESTs stored in Genbank of date

# slaves	100 MBit/s complete runtime	speedup	1000 MBit/s complete runtime	speedup
0 (serial)	13360 sec	1	13360	1
2	7738 sec	1.73	6352	2.10
3	6264 sec	2.13	5011	2.57
4	5263 sec	2.54	4215	3.17
5	4785 sec	2.79	3741	3.57
6	4392 sec	3.04	3410	3.92

Table 1: Runtime and speedup for MPI-ClustDB results of detecting all common substrings of length $M = 50$ in all human ESTs considering two network velocities.

2005-04-06. The task is the identification of all common substrings of length 50 in the test set. The serial ClustDB programm needs a total of 3 hours and 42 minutes to solve the problem of detecting all 7,059,622 substring clusters of match length 50 for all human ESTs. Table 1 shows how the runtime of MPI-ClustDB alters for employing different numbers of CPUs. The complete runtime decreases for any addition of a CPU in the cluster leading to an overall runtime of 1 hours and 15 minutes for 7 personal computers and a 100 MBit/s network. Using the gigabit connection the runtime decreases to 56 minutes. Thus we are approximately four times faster with MPI-ClustDB than with the serial ClustDB software. Figure 2 analyses the reasons of the performance gains. The bisecting line presents the optimal speedup. Ideally parallel computing using p processors should be p times faster than the serial program. The left plot displays the achieved speedup for a 100 MBit/s network. Employing only one slave increases time of computation. But for at least two slaves we see a sound speedup for the parallelization of the Substring Detection step (square symbol). The other two parallel steps Start Word Sort (diamond) and Output Conversion (triangle) do not scale well. This results from an excessive amount of overhead that is due to communication among the processes. The amount of data that has to be distributed in these parts is comparatively large and the period for sending the data to another process is out of scale compared to the calculations performed on the data. The right plot in Fig. 2 shows the results for a 1000 MBit/s network connection. A significant speedup for each step is achieved resulting in a greater overall speedup. Compared to the slower network, the speedup for step 1 increases best while the speedup for step 2, that already scaled well for 100 Mbit/s, improves just slightly. The reason is that the time consumption for step 2 is mainly due to computation and not communication.
To account for diverse network velocities, our program optionally utilizes parallel computing for Substring Detection only (overall speedup 1) or with all three parts being parallelized (overall speedup 2). The two resulting speedups are displayed in Fig. 2. For the slower network connection overall speedup 1 is superior to overall speedup 2. Although step 1 and 3 scale poorly for the slower network, overall speedup 1 is just slightly better. This results from the fact that the Substring Detection step takes about 83% of the overall CPU time. By contrast omitting the parallelization of part 1 and 3 leads to a notedly larger runtime for the 1000 MBit/s ethernet.

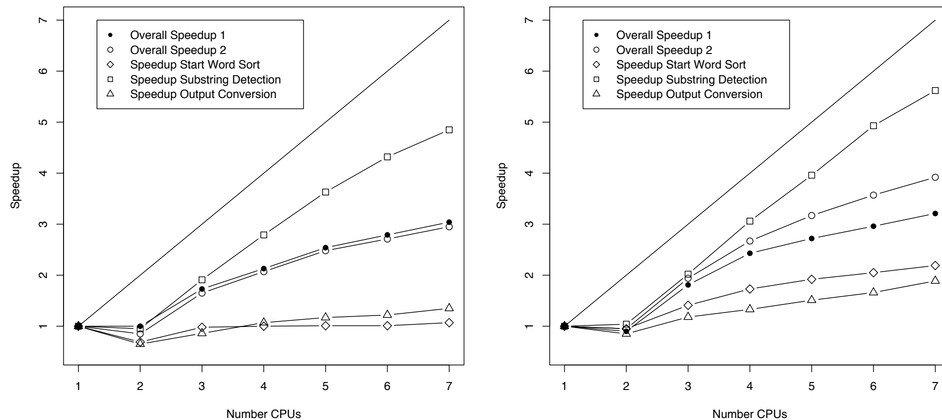

Figure 2: Speedup, as a function of the number of processors, for a 100 (left) and a 1000 (right) MBit/s network connection. The task is detecting all common substrings of length $M = 50$ in all human ESTs. Speedup 1 displays results for only Substring Detection being parallelized and speedup 2 displays results with parallel computation of all three algorithmic steps.

5 Discussion and Summary

Nowadays Bioinformatics in general and sequence comparison in particular is faced with very large datasets. ClustDB, our tool for finding common substrings in DNA-sequences, is able to work on a greater amount of data than the competing programs. Additionally we achieved to significantly reduce the runtime via our parallel implementation MPI-ClustDB using a relatively inexpensive PC cluster. We parallelized the three most time consuming parts of the programm, but for a 100 MBit/s network connection only one part shows a speedup and is therefore used in it‘s parallel implementation. Increasing the network speed to 1000 MBit/s yields significant speedups for all parallelized parts and cleary an ascending overall speedup.
The parallel computation of each of the three parts is a problem of allocating independant tasks to processors. The goal is to execute the tasks as quickly as possible. In the Output Conversion step the fixed size chunking strategie is used that is theoretically preferable compared to the others. But due to sundry constraints fixed size chunking is not applicable in steps one and two. For example in the first parallel part N could be enlarged to reduce processor idle time. But that would lead to a larger number of files to be read from in the next step and overall runtime was observed to increase. Nevertheless more sophisticated scheduling strategies may be possible and will be analysed in further developements.
We will investigate and optimize the scaleability of MPI-ClustDB next by running it with a greater number of processors. Furthermore we will try to parallelize additional parts of ClustDB. First aims are the calculation of sequence clusters (a subset of sequences having no substring of length M in common with any sequence outside the subset) derived from the substring clusters and extending pairs of matching substrings with errors. Based on

MPI-ClustDB a parallel solution for 64 bit shared memory systems is intended afterwards.

Acknowledgment

This project was supported by the BMBF Germany under contract number 0312705A. The authors would like to thank Friedrich Möller for technical assistance.

References

[AKO04] M.I. Abouelhoda, S. Kurtz, E. Ohlebusch. "Replacing Suffix Trees with Enhanced Suffix Arrays" Journal of Diskrete Algorithms, No. 2, 53-86, 2004.

[De02] A.L. Delcher, A. Philippy, J. Carlton, S.L. Salzberg. "Fast algorithms for large scale genome alignment and comparison", Nucleic. Acids Research, Vol. 30, No. 11, 2002.

[FAK98] N. Futamura, S. Alura, S. Kurtz. "Parallel Suffix Sorting", Proc. 9th International Conference on Advanced Computing and Communications, 76-81, 2001.

[GSN98] W. Gropp, M. Snir, B. Nitzberg, "MPI: The Complete Reference", 2nd edn, MIT Press, Cambridge, MA, 1998.

[GTL01] W. Gropp, R. Thakur, E. Lusk. "Using MPI-2", MIT Press, Cambridge, MA, 2001.

[Ha97] T. Hagerup. "Allocating independent tasks to parallel processors: an experimental study", J. Parallel Comput., Vol. 47, 185-197, 1997.

[IK02] C. S. Iliopoulos, M. Korda. "Massively Parallel Suffix Array Construction", Proc. 25th Conference on Current Trends in Theory and Practice of Informatics, 371 - 380, 1998.

[Ka03] A. Kalyanaraman, S. Alura, S. Kothari, V. Brendel. "Efficient clustering of large EST data sets on parallel computers", Nucleic Acid Research, Vol. 31, No. 11, 2963-2974, 2003.

[KMW06] J. Kleffe, F. Möller, B. Wittig. "ClustDB: A high performance tool for large scale sequence matching", Proceedings DEXA 2006.

[Ku04] S. Kurtz, A. Philippy, A.L. Delcher, M. Smoot, M. Shumway, C. Antonescu, S.L. Salzberg. "Versantile and Open Software for Comparing Large Genomes", Genome Biology, 5 (R12), 2004.