Knowledge Processing and Computer Architecture

Omerovic, S., Tomazic,S., Milovanovic, M., and Torrents, D.

Abstract— ***This position paper argues that the most suitable computer architecture for knowledge processing in bioinformatics is TM (transactional memory), ported into the DSM (distributed shared memory) environment, and expanded with elements of SMT (simultaneous multithreading. Current implementations of TM are in the SMP (shared memory multiprocessor) environment and without extensive support for SMT. In order to justify this position, the paper treats the field of decision making (DM) applied to knowledge processing for the need of bioinformatics. The basic idea is to have an automated reasoning mechanism Decision Making System (DMS) able to make a decision (related to the corresponding question), if the input data are in a text form (like it is the case in genomic processing). An illustration of Data modelling and Analysis layer, as a part of DMS and for the purpose of genomic processing, is given next. Bioinformatics experts mostly use BLAST software output in order to make decisions concerning genomic data. This DM process is mostly done manually, making it dependent on the expert knowledge and talent, in a way which is (for the most part) not automated and therefore not uniform and not eligible for global data exchange and comparing. We have proposed an approach that may lead to automatization of the DM process, based on the theory of concept processing (upgrade of Data Mining and Semantic Web). Analysing the typical processing needs in the set of genomic data (atomic access and high levels of parallelism), conclusion is that TM Systems (TMS) offer the processing capabilities demanded by both the Data Modelling and Analysis (layer 2) and Concept Processing (layer 3), but first have to be ported from SMP to DSM and enhanced with SMT (to support the higher levels of parallelism), before they can be successfully applied in genomic processing and other areas of science where huge-volume knowledge processing through DM is required.***

Index Terms— *genomic, knowledge, memory, modelling*

Manuscript received January 30, 2007. This work was supported in part by the Supercomputing Centre, Barcelona, Spain.

Sanida Omerovic and Saso Tomazic are with the Faculty of Electrical Engineering, University of Ljubljana, Slovenia (e-mail: sanida.omerovic@lkn1.fe.uni-lj.si, saso.tomazic@fe.uni-lj.si). Milos Milovanovic is with Supercomputing Centre, Barcelona, Spain (e-mail: milos.milovanovic@bsc.es). David Torrents is with the ICREA-Supercomputnig Centre, Barcelona, Spain (e-mail: david.torrents@bsc.es).

1. Introduction to Decision Making Systems

Every day surrounds us a vast amount of data: newspapers, radio, TV, Internet, etc. And every day a person makes hundreds of decisions: what to eat, what to wear, where to go, who to contact, etc. The whole DM process is happening person's head, so the most general and oldest DMS is human brain. Decisions are made by input data (based on person's everyday perception), person's DM criteria's (already stored in person's brains based on life experience) and predefined knowledge (everything what person have learned from birth up to the present moment).

Every system that has unstructured data and question/s as input and decision/s as output, can be observed as a DMS consisting of the following four layers: Data retrieval layer + Data modeling and Analysis layer + Concept processing layer + Decision making layer. For each one of these layers, have its organization presented, and discussed its processing needs (implementation requirements). Observed DMS for the purpose of DM in the machine world has the analogy of the human DM process mentioned above. DMS layers are presented in Figure 1 and one can conclude that these layers are involved iteratively.

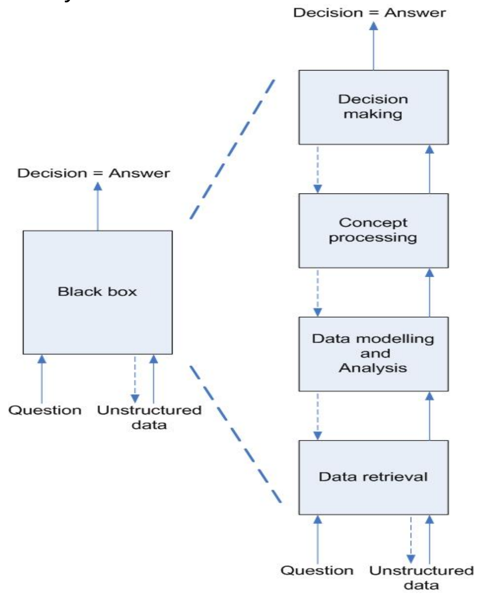


Figure 1. DMS – from general to layered view. Inputs are question and unstructured data, and output is decision (usually in a form of answer to the specific question). DMS core consists of the following four layers: Data retrieval, Data modelling and Analysis, Concept processing, and Decision making.

In the case of the Data retrieval (DR) layer, the essence is (as the name itself says) retrieval of unstructured data. So, in this layer one is gathering all types of data (text, audio, video, and pictures) into DMS. Here, one is dealing with a different data sources, and filtering helps extract only the data needed for Decision.

In the case of the Data modelling and analysis (DMA) layer, the essence is modelling and analysis of the unstructured data. So, after gathering as much as possible data related only to output Decision, data are modelled in a uniform manner, so that they can be compatible for the further analysis. At this point, one tries to eliminate noisy data (data that may be invalid), so that only valid data are used to make concepts from, which is next step.

In the case of the Concept processing (CP) layer, the essence is that it includes two sub-layers: Concept Modelling and Concept Search. This is the core of proposed DMS, and the idea is that reasoning mechanisms for Concept definition, Concept population, and Concept replacement are embedded into this level. In this way, knowledge is presented by concepts, and DMS operates on Conceptual-level, instead of Semantic-level like search engines today (Google, Yahoo, etc). This can be done by using Neural Networks, Fuzzy logic, Space Vector Model [1], or similar statistical methods.A detailed idea of CP layer is presented in Section 3.

In the case of the DM layer, the essence is that this layer contains a reasoning mechanism that combines concepts (from below layer) and DM criteria's that are stored in this layer and directly related only to output Decision. DM criteria are defined by the party that is using DMS (it can be a person or a company).

2. *Data Modeling and Analysis for Genomic Processing*

This section gives practical example of the first two DMS layers, namely DR and DMA, in the case of genomic processing. As mentioned before, in a DMS, input can be any kind of unstructured data. For the purpose of this section, analysis is limited on text only, because genomic processing uses only text as input. That fact serves to us as a justification to apply DMS in genomics.

Genomic researchers mostly deal with similarity issues between genomic sequences. Genomic sequences are treated as long sequences of letters A (Adenine), G (Guanine), C (Cytosine), and T (Thymine) which represents nitrogenous bases in protein structure.

As a illustration for previous sentence, DNA nitrogenous base pairs and a DNA sequence example is presented in Figure 2 and Figure 3, respectively

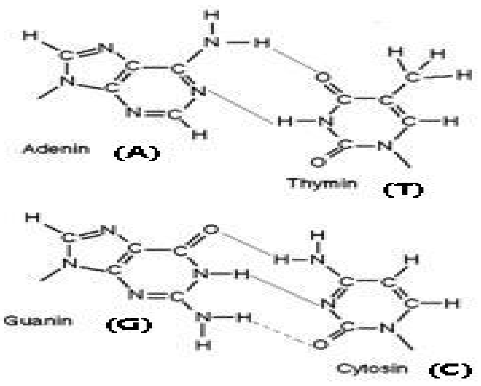
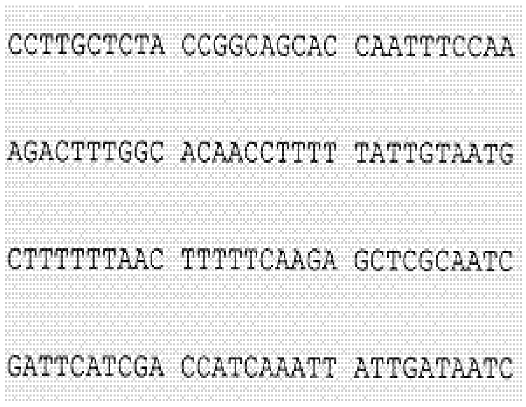


Figure 2. DNA nitrogenous base pairs



CCTTGCTCTA CCGGCAGCAC CAATTTCCAA

AGACTTTGGC ACAACCTTTT TATTGTAATG

CTTTTTTAAC TTTTTCAAGA GCTCGCAATC

GATTCATCGA CCATCAAATT ATTGATAATC

Figure 3. A DNA sequence presented as an array of letters which are mapping the nucleotides in DNA (consisted of one of four types of nitrogenous bases A/G/C/T, a five carbon sugar, and molecule of phosphoric acid).

Observing through the DMS model described above, genomic processing moves automatically to the DMA layer. Obtaining the available genomic sequences (basically DR layer) can be done by Internet for no cost in academic purposes. One can download the genomic sequences (of a human, chimpanzee, mouse, etc) from web pages like: www.ensembl.org, www.ncbi.nlm.nih.gov, genome.ucsc.edu, etc.

For the purpose of the DMA layer, genomic experts use software that is able to find similar patterns (words) within the long genomic segment.

Examples of this software are BLAST [2] (the mostly frequently used software), Smith Waterman [3], FastA [4], and others. These programs find similarity between a query sequence and the sequences within the database.

In the example shown at Figure 4 and Figure 5, one can see a fraction of the results obtained from a BLAST comparison of protein SLC7A7 (human) against a SwissProt (http://www.isb-sib.ch) database of proteins. Two illustrative examples that show from a perfect (word) mach to a similar mach are presented.

```
>gi|12643348|sp|Q9UHI5|LAT2_HUMAN
<http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=Protein&list_uids=12643348&dopt=GenPept> Gene info
< http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=gene&cmd=search&term=12643348%5BPUID%5D> Large neutral amino
acids transporter small subunit 2 (L-type  amino acid transporter 2) (hLAT2)
Length=535
Score =  665 bits (1717),  Expect = 0.0, Method: Composition-based stats.
Identities = 332/332 (100%), Positives = 332/332 (100%), Gaps = 0/332 (0%)

Query  1     MGIVQICKGEYFWLEPKNAFENFQEPDIGLVALAFLQGSFAYGGWNFLNYVTEELVDPYK  60
MGIVQICKGEYFWLEPKNAFENFQEPDIGLVALAFLQGSFAYGGWNFLNYVTEELVDPYK
Sbjct  204    MGIVQICKGEYFWLEPKNAFENFQEPDIGLVALAFLQGSFAYGGWNFLNYVTEELVDPYK  263

Query  61    NLPRAIFISIPLVTFVYVFANVAYVTAMSPQELLASNAVAVTFGEKLLGVMAWIMPISVA  120
NLPRAIFISIPLVTFVYVFANVAYVTAMSPQELLASNAVAVTFGEKLLGVMAWIMPISVA
Sbjct  264    NLPRAIFISIPLVTFVYVFANVAYVTAMSPQELLASNAVAVTFGEKLLGVMAWIMPISVA  323

Query  121  LSTFGGVNGSLFTSSRLFFAGAREGHLPSVLAMIHVKRCTPIPALLFTCISTLLMLVTSD  180
LSTFGGVNGSLFTSSRLFFAGAREGHLPSVLAMIHVKRCTPIPALLFTCISTLLMLVTSD
Sbjct  324    LSTFGGVNGSLFTSSRLFFAGAREGHLPSVLAMIHVKRCTPIPALLFTCISTLLMLVTSD  383

Query  181  MYTLINYVGFINYLFYGVTVAGQIVLRWKKPDIPRPIKINLLFPIIYLLFWAFLLVFSLW  240
MYTLINYVGFINYLFYGVTVAGQIVLRWKKPDIPRPIKINLLFPIIYLLFWAFLLVFSLW
Sbjct  384    MYTLINYVGFINYLFYGVTVAGQIVLRWKKPDIPRPIKINLLFPIIYLLFWAFLLVFSLW  443

Query  241  SEPVVCGIGLAIMLTGVPVYFLGVYWQHKPKCFSDFIELLTLVSQKMCVVVYPEVERGSG  300
SEPVVCGIGLAIMLTGVPVYFLGVYWQHKPKCFSDFIELLTLVSQKMCVVVYPEVERGSG
Sbjct  444    SEPVVCGIGLAIMLTGVPVYFLGVYWQHKPKCFSDFIELLTLVSQKMCVVVYPEVERGSG  503

Query  301  TEEANEDMEEQQQPMYQPTPTKDKDVAGQPQP  332
TEEANEDMEEQQQPMYQPTPTKDKDVAGQPQP
Sbjct  504    TEEANEDMEEQQQPMYQPTPTKDKDVAGQPQP  535
```

Figure 4. BLAST Sample session, perfect match. Comparison of protein SLC7A7 (human) against the same protein.

```
>gi|12643378|sp|Q9UM01|YLA1_HUMAN
   <http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=Protein&list_uids=12643378&dopt=GenPept> Gene info
   <http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=gene&cmd=search&term=12643378%5BPUID%5D> Y+L amino acid
    transporter 1 (y(+)L-type amino acid transporter
    1) (y+LAT-1) (Y+LAT1) (Monocyte amino acid permease 2) (MOP-2)
    Length=511
   Score =  257 bits (656),  Expect = 4e-68, Method: Composition-based stats.
   Identities = 138/315 (43%), Positives = 203/315 (64%), Gaps = 10/315 (3%)

   Query  2       GIVQICKGEYFWLEPKNAFENFQEPDIGLVALAFLQGSFAYGGWNFLNYVTEELVDPYKN  61
                  GIV++ +G    E  N+FE     +G +ALA     F+Y GW+ LNYVTEE+ +P +N
  Sbjct  202      GIVRLGQGASTHFE--NSFEG-SSFAVGDIALALYSALFSYSGWDTLNYVTEEIKNPERN  258

  Query  62      LPRAIFISIPLVTFVYVFANVAYVTAMSPQELLASNAVAVTFGEKLLGVMAWIMPISVAL  121
                 LP +I IS+P+VT +Y+  NVAY T +  +++LAS+AVAVTF +++ G+  WI+P+SVAL
  Sbjct  259      LPLSIGISMPIVTIIYILTNVAYYTVLDMRDILASDAVAVTFADQIFGIFNWIIPLSVAL  318

  Query  122    STFGGVNGSLFTSSRLFFAGAREGHLPSVLAMIHVKRCTPIPALLFTCISTLLMLVTSDM  181
                S FGG+N S+  +SRLFF G+REGHLP  + MIHV+R TP+P+LLF  I  L+ L   D+
  Sbjct  319     SCFGGLNASIVAASRLFFVGSREGHLPDAICMIHVERFTPVPSLLFNGIMALIYLCVEDI  378

  Query  182    YTLINYVGFINYLFYGVTVAGQIVLRWKKPDIPRPIKINLLFPIIYLLFWAFLLVFSLWS  241
                + LINY  F  + F G+++ GQ+ LRWK+PD PRP+K+++ FPI++ L   FL+   L+S
  Sbjct  379      FQLINYYSFSYWFFVGLSIVGQLYLRWKEPDRPRPLKLSVFFPIVFCLCTIFLVAVPLYS  438

  Query  242    EPVVCGIGLAIMLTGVPVYFL--GVYWQHKPKCFSDFIELLTLVSQKMCVVVYPEVERGS  299
                  + +   IG+AI L+G+P YFL   V    +P      +   T   Q +C+ V  E++
  Sbjct  439      DTINSLIGIAIALSGLPFYFLIIRVPEHKRPLYLRRIVGSATRYLQVLCMSVAAEMDLED  498

 Query  300    GTEEANEDMEEQQQP  314
               G E     M +Q+ P
 Sbjct  499     GGE-----MPKQRDP  508
```

Figure 5. BLAST Sample session, similar match. Comparison of protein SLC7A7 (human) against a SwissProt database of proteins.

As it is shown in the above two figures, BLAST expresses the level of similarity between query sequence and database sequence in terms of: score, expectations, method, identities, positives, and gaps. Here is where proposed DMA layer is finishing, and from this point inferring needs to be done by genomic experts on the bases of software (ex. BLAST) output, and knowledge gathered elsewhere (brains, book, computers, etc).

The complete philosophy of sequence comparison in biological context relies on the fact that similar sequences have similar functions. Therefore, comparative analysis of sequences help researchers infers possible functions, which guides them for further molecular analysis.

The possible interpretations of this type of comparative analysis are very wide and depend very much on initial question. Following authors initial scheme (Figure 1), one could see this step as the core of building the CP layer. This topic is further discussed in Section 3.

Also, a forthcoming challenge in the field of comparative genomic analysis is to compare large amounts of genomic data (letters). Current databases are already reaching size limits that make simple comparisons not possible. These limitations are probably due to lack of memory that could be eventually solved at hardware level or by modifying the structure of data to make them more efficient for processing. For example, if one wants to compare one mammalian genomic sequence against all existing mammalian sequences, one would need a database with memory storage of 60 GB. Every day, researchers are producing more and more genomic sequences. Scientific community expects a large amount of genomic data coming from meta-genomic (Environmental genomic) projects like Sargasso Sea Project [5]. In general, if one wants to have complete and accurate results in the domain of knowledge extraction, big database is an advantage. That is the problem that can be solved with TM, which is explained more in Section 4

3. Concept Processing

State of the art Knowledge-retrieval systems are based on Semantic retrieval, but Knowledge-retrieval systems will become much more efficient once they start using CP. If one says “I am married” and “I have a husband” – these two statements are semantically different, but they both refer to the same concept. If the retrieval is based on semantics, only a subset of knowledge will be retrieved from the database or the Internet. If the retrieval is based on the concepts, all the relevant knowledge that points to the same concept will be retrieved.

3.1. Internal structure - Core Concept and Onion-layers

One important research problem is how to represent concepts. A trivial solution (which does not make sense for large sets of data) is to come up with a huge Case Statement that will include all semantic structures that lead to the same concept. This is a brute force approach, and can be applied only to limited vocabulary problems, like the processing of patent data [6], or similar. The only realistic solution is to employ a CP software architecture, based on Concept Networks, expanded into Concept Web, using a modular Onion-layered type structure, as indicated in Figure 6.

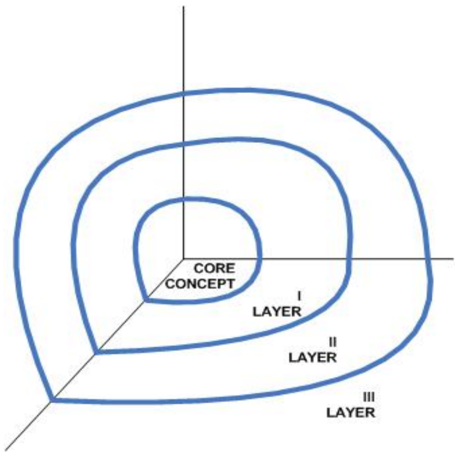
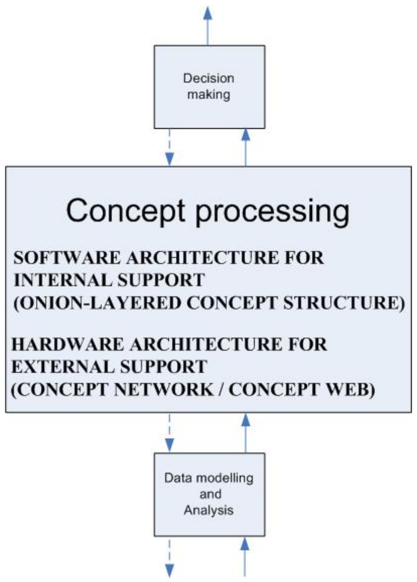


Figure 6. Concept internal organization - Onion-layered structure. The main idea is to have core of the concept in which minimum data is stored for basic concept understanding. Here, in the core, should be stored the essence of a concept, so that when a person not familiar with the topic processes it, it is able to understand it. Like, when learning completely new things, first one has to be aware of the essence in order to understand anything related to the topic/concept involved. Next layer (I LAYER) gathers a set of concepts related to the observed CORE CONCEPT. So, if one is not able to understand the core itself, then one moves to LAYER I, which contains other concepts related to the observed concept, so one have more knowledge and therefore more ability to understand it. If the amount of knowledge stored in the LAYER I is still not enough for understanding the CORE CONCEPT then one move to LAYER II having on disposal even more concepts related to the observed CORE CONCEPT. And so on for LAYER III…

Onion-layered type structure is similar to the process of learning. When one is not able to learn from the essence of the matter presented, one searches for more data (like the examples or relations to the other topics) in order to understand it. CORE CONCEPT represents the essence of the matter presented and examples and relatedness to other topics is what LAYER I / II / III / etc are presenting.

Basically, if one observes CORE CONCEPT as a sphere, I LAYER is a sphere with a bigger radius containing CORE CONCEPT; LAYER II is a sphere containing both LAYER I and CORE CONCEPT; LAYER III is a sphere containing LAYER II, LAYER I, and CORE CONCEPT, and so on….

The CP software architecture, with an indication that its efficient execution implies the existence of the underlying computer architecture that represents a perfect concept match for the processing needs, is presented in Figure 7. The CORE CONCEPT includes the essence of the concept definition, and outer layers represent concept refinements.



Decision making
Concept processing
SOFTWARE ARCHITECTURE FOR INTERNAL SUPPORT (ONION-LAYERED CONCEPT STRUCTURE)
HARDWARE ARCHITECTURE FOR EXTERNAL SUPPORT (CONCEPT NETWORK / CONCEPT WEB)
Data modelling and Analysis

Figure 7. Software and Hardware demands for CP layer. Software architecture in CP layer should support Onion-layered concept structure, and hardware architecture should support Concept Network/Concept Web structure.

3.2. External structure – Concept Network and Concept web

Now, the question arises how to organize CORE CONCEPTS (or just short CONCEPTS) among each other. Two solutions are proposed: CONCEPT NETWORK where concepts are related with one directional arc that has verb attached to it (original idea taken from RDF ontology language - www.w3.org/**RDF**/), and CONCEPT WEB, which is the extension of CONCEPT NETWORK, where relations between CONCEPTS are also CONCEPTS.

Figure 8 and Figure 9 show examples of CONCEPT NETWORK and the related CONCEPT WEB in it's the simplest form. The former includes the nodes that refer to subject and object matter, while the predicate matter is referred to as arcs. The later is derived from the former by promoting the predicate arcs into nodes, using a generalization approach (verbs of the arcs are converted into the nouns of the nodes). Subjects and objects are observed as core concepts, and verbs are observed as relations/core concepts respectively.

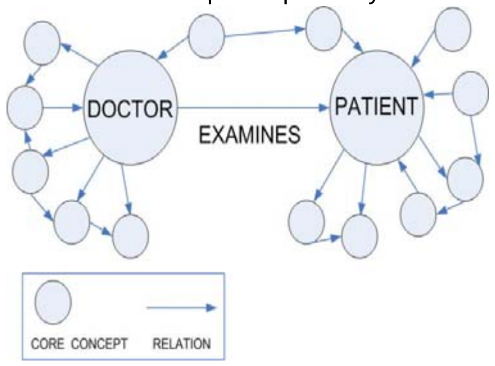


Figure 9. Concept network. The core concepts DOCTOR and PATIENT contain subject matter related to those two concepts. For example: DOCTOR = a person trained in the healing arts and licensed to practice. PATIENT = one who receives medical attention, care, or treatment.

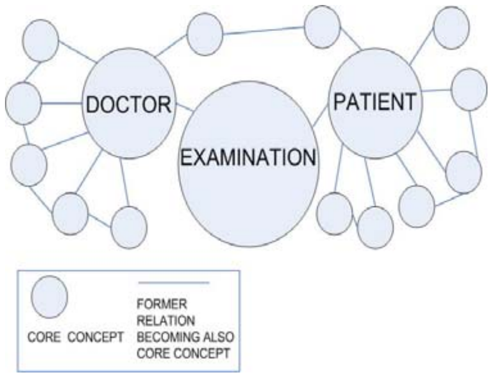


Figure 10. Concept web. The relations are also core concepts. So, besides core concepts DOCTOR and PATIEN, core concept EXAMINATION is added = a medical inquiry into a patient's state of health.

Programs already do exist (e.g., OntoLearn [6]) that build concept network (sometimes called semantic nets) using the lexicosemantic relations like: Hyponym (a word or phrase whose semantic range is included within that of another word), Hypernom (opposite of a hyponym), Gloss (concept appears in the definition of another concept), Topic (concept often co-occurs with another concept), etc.

CP consisted of concepts and their relations which are built on semantic bases is avoided in proposed DMS, because that brings CP which is language and grammar independent. Idea is to have processing of the conceptual level, with the words that have unique meaning (for that purpose genomic processing is ideal because letters A, C, G, T in genomic do have unique meaning).

The move from semantics to concepts implies the activities that (in their simplest form) convert the above mentioned semantic web into a related concept web.

4. Computer Architecture Considerations

Knowledge is often ambiguous and therefore not scalable and not suitable for further processing. In most of the cases, user is provided with a huge amount of data, but without any possibility for automatic logical reasoning on the top of those data. Concept processing based on transactional memory (extended into DSM and expanded with SMT) as a possible solution to overcome this problem. Actually, TM [7] is the solution for a wide variety scientific of supercomputing problems not discussed in this paper.

All problems discussed so far use concepts as atoms of knowledge (which fits into the atomic transaction structure of TM). Concept definitions in their atomic form directly map on the TM constructs, and concept organization presented in this paper goes in two directions: one related to application issues, and the other related to technology issues.

4.1. Application Issues

The relationship between application demands and constructs offered by the underlying architecture is essential for the required "perfect match" of the application and the architecture. No matter if one talk about CP for Internet oriented DM in business (to detect potential profit strategies), or about genomic processing in distributed database (to detect potential genetic developed diseases), application requirements can be described as follow:

```
start atomic transaction
  access a data structure
  perform the related processing
  detect potential hazards in the
  system
  commit or rollback
end atomic transaction
```

This very same computational structure is built into a typical TMS, and can be directly translated into the code for a TMS, as indicated in the following example:

This computational structure is built into a typical TM like system and can be directly transformed into the proper TM form. This transformation will be presented using AMMP application [9]. AMMP is a modern full-featured molecular mechanics, dynamics and modelling program. It can manipulate both small molecules and macromolecules including proteins, nucleic acids and other polymers. This application is also part of Spec OMP 2001 benchmark [9] for testing platforms for execution of the parallel application. The idea is to make TM completely transparent from the researcher/programmer because in that way all difficulties of writing parallel applications are hidden and researcher productivity is much better. Idea is that "useful" code should be just surrounded into atomic block and everything else should be as it was before. After that, specialized compiler should transform the code. Original code and the transformed code are presented in Figure 11 and Figure 12 respectively.

```
atomic {
  ux = (a2->dx -a1->dx)*lambda
        +(a2->x -a1->x);
  uy = (a2->dy -a1->dy)*lambda
        +(a2->y -a1->y);
  uz = (a2->dz -a1->dz)*lambda
        +(a2->z -a1->z);
  r = one/( ux*ux + uy*uy +
      uz*uz);
  r0 = sqrt(r);
  ux = ux*r0;
  uy = uy*r0;
  uz = uz*r0;
  k = -dielectric*a1->q*a2->q*r;
  r = r*r*r;
  k = k + a1->a*a2->a*r*r0*six;
  k = k - a1->b*a2->b*
      r*r*r0*twelve;
  a1fx = a1fx + ux*k;
  a1fy = a1fy + uy*k;
  a1fz = a1fz + uz*k;
  a2->fx = a2->fx - ux*k;
  a2->fy = a2->fy - uy*k;
  a2->fz = a2->fz - uz*k;
}
```

Figure 11. Critical part of the AMMP application which should be executed atomically. This code presents standard C/C++ code and with the simple surrounding that code into atomic block one have the TM application. This way TM mechanism is completely transparent for the researcher, it hides the difficulties of writing the parallel applications and extends the researchers productivity.

```
{ startTransaction(); {
   write(t, &ux, (*read(t,&((
    *read(t, &a2))->dx)) -
    *read(t, &((*read(t, &a1))->dx))
)* *read(t, &lambda) +
    (*read(t, &((*read(t, &a2))-
>x))-
    *read(t, &((*read(t, &a1))-
>x))));
   write(t, &uy, ( *read(t, &( (
*read(t, &a2)  ) ->dy) ) -
    *read(t, &( ( *read(t, &a1)  ) -
>dy) ) )* *read(t, &lambda) +
    (*read(t, &( ( *read(t, &a2)  )
->y) ) -
    *read(t, &( ( *read(t, &a1)  ) -
>y) ) ));
   write(t, &uz, ( *read(t, &( (
*read(t, &a2)  ) ->dz) ) -
    *read(t, &( ( *read(t, &a1)  ) -
>dz) ) )* *read(t, &lambda) +
    ( *read(t, &( ( *read(t, &a2)  )
->z) ) -
    *read(t, &( ( *read(t, &a1)  ) -
>z) ) ));
   write(t, &r,  *read(t, &one) /(
*read(t, &ux) * *read(t, &ux) +
    *read(t, &uy) * *read(t, &uy) +
*read(t, &uz) * *read(t, &uz) ));
   write(t, &r0, sqrt( *read(t, &r)
));
   write(t, &ux,  *read(t, &ux) *
*read(t, &r0) );
   write(t, &uy,  *read(t, &uy) *
*read(t, &r0) );
   write(t, &uz,  *read(t, &uz) *
*read(t, &r0) );
   write(t, &k, - *read(t,
&dielectric) *
    *read(t, &( ( *read(t, &a1)  ) -
>q) ) *
    *read(t, &( ( *read(t, &a2)  ) -
>q) ) * *read(t, &r) );
   write(t, &r,  *read(t, &r) *
*read(t, &r) * *read(t, &r) );
   write(t, &k,  *read(t, &k) +
*read(t, &( ( *read(t, &a1)  ) ->a))
*
    *read(t, &( ( *read(t, &a2)  ) -
>a) ) *
    *read(t, &r) * *read(t, &r0) *
*read(t, &six) );
   write(t, &k,  *read(t, &k) -
*read(t, &( ( *read(t, &a1)) ->b) )
*
    *read(t, &( ( *read(t, &a2)  ) -
>b) ) *
    *read(t, &r) * *read(t, &r) *
*read(t, &r0) * *read(t, &twelve) );
   write(t, &a1fx,  *read(t, &a1fx)
+ *read(t, &ux) * *read(t, &k) );
   write(t, &a1fy,  *read(t, &a1fy)
+ *read(t, &uy) * *read(t, &k) );
   write(t, &a1fz,  *read(t, &a1fz)
+ *read(t, &uz) * *read(t, &k) );
   write(t, &((*read(t, &a2)) ->fx),
*read(t,&((*read(t, &a2)) ->fx))-
    *read(t, &ux) * *read(t, &k) );
   write(t, &( ( *read(t, &a2)  ) -
>fy ) ,
    *read(t, &((*read(t, &a2))->fy))
- *read(t, &uy) * *read(t, &k) );
   write(t, &( ( *read(t, &a2)  ) -
>fz ) ,
    *read(t, &( ( *read(t, &a2)  ) -
>fz) ) - *read(t, &uz) *
    *read(t, &k) );
  } endTransaction();
}
```

Figure 12. Generated code for the part of the AMMP application presented in Figure 11). This is generated code suitable for STM (Software transactional memory). It presents how difficult it would be to write program without support of the external tools. The idea is to make TM completely transparent for the researcher and to create external tools which will provide support for the transactional memory. Basic support for the STM as a Proof of Concept is the next step together with research in Hardware TM and Hybrid TM in order to produce high-performance applications.

In this way one can have a very user friendly system, because programmer should just surround critical code into atomic locks. Also, system will have high performances because it can be ported for HTM (Hardware Transactional memory) of HyTM (Hybrid Transactional Memory), which will speedup the execution of the applications. Next step is to research and develop specialized hardware that will accelerate our TM applications. In that way one can have a high-performance system for extraction of knowledge.

However, since TM is defined in the SMP environment, and researchers in Genomic processing report the need for huge and distributed memory and databases, it has to be ported to DSM and expanded with SMT, before it can be used in applications of interest for this work.

A good source of information about DSM is the survey paper [11] For the DSM and SMT extensions to be applicable both to the second and the third DM layers presented here, the development has to take into consideration the needs of both, applications like BLAST and algorithms like those suggested in CP layer.

4.2. Technology Issues

The TMS architecture has been designed for SMP paradigm, and its potentials for parallel processing are limited. Typical number of nodes on a SMP bus is equal to 16, 32, or 64, and further expansions are not possible, due to current technology limitations (current bus speed is not large enough to enable a larger number of processors). On the other hand, amount of parallelism involved in the above discussed applications is enormous, and may be of the order of 1M, 2M, or even 4M.

Fortunately, with introduction of optics into the domain of system communications, in the future one can expect that the size of SMP systems can grow beyond 64, 128, or 256. Also, if the TMS concept is ported from SMP to DSM (where the current levels of parallelism are 1K, 2K, or 4K), then parallelisms above 1M become readily available, and the transactional memory paradigm can find its usage in complex systems oriented to concept modeling for bio-informatics.

5. Generalized Structure of a System for Concept Processing

With all above in mind, proposed structure of a future Knowledge Processing System is as indicated in Figure 13. The bottom, Computer Architecture, layer is consisted of two sub layers: TMS/DMS environment layer and appropriate Operating System (OS) and Optimizing Compilers layer. On the top of the bottom layer (including both sub layers) works Knowledge Understanding layer that helps build the semantics, and even more importantly, the concept models, used to extract knowledge for the applications of interest. Finally, the top layer, called Application layer, is related to applications like Internet search, genomic processing, etc (all based on text as the input data).

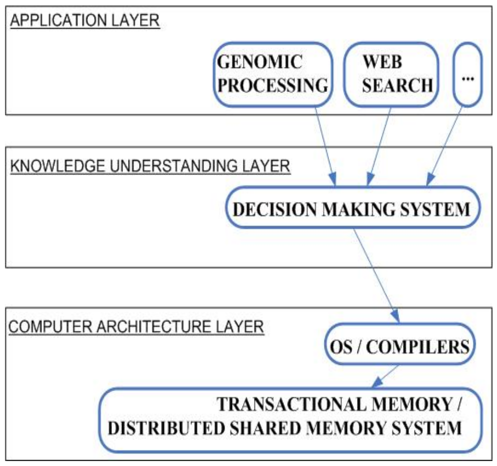

APPLICATION LAYER
GENOMIC PROCESSING
WEB SEARCH
...
KNOWLEDGE UNDERSTANDING LAYER
DECISION MAKING SYSTEM
COMPUTER ARCHITECTURE LAYER
OS / COMPILERS
TRANSACTIONAL MEMORY / DISTRIBUTED SHARED MEMORY SYSTEM

Figure 13. Knowledge Processing System: Computer Architecture layer, DMS layer, Application layer

A possible strategy leading to design and implementation of a system from Figure 10 implies the following steps:

- Prior to application, experts in fields like Knowledge processing (in general) or Genomic processing (specific), are consulted to see what are their typical problems and what are the typical computational patterns involved.
- Appropriate algorithms are developed, which are oriented to fast execution on TMS architectures. If needed, the architecture of the underlying machine can be modified.
- Software is developed that converts the existing tools into forms that can run efficiently on the TMS architecture.
- Performance is measured, and possibly some of he software constructs are ported into hardware.

Of course, once the system is designed, and lessons are learned from the deployment of the system, after the incubation period is lived through, ideas will be generated on how one can further improve the speed and other important aspects of the processing system involved.

6. Conclusion

In this paper, we have presented an architectural and algorithmic support for a Knowledge Processing System in selected high-demand applications. One possible scenario implies a three layer system:

- Top: Application (like Genomic Processing).
- Medium: Knowledge Understanding, based on Data Mining, Semantic Web, and CP (as the most sophisticated approach).
- Bottom: Computer Architecture, along the concepts of SMP and DSM, with a special emphasis on TM), which uses a computational paradigm compatible with the needs of CP.

Acknowledgement

The authors would like to thank Prof. Pavle Andjus, Institute for Physiology and Biochemistry School of Biology; University of Belgrade; Serbia, for his useful comments.

References

[1] Salton, G., Wong, A., "A Vector Space Model for Automatic Indexing," Communications of the ACM, 1975, pp. 613 - 620 Vol. 18, Issue 11.

[2] BLAST, December 2006. Available online: www.ncbi.nlm.nih.gov/genome/seq/BlastGen/BlastGen.cgi?taxid=9606

[3] "Identification of Common Molecular Subsequences," Temple F. Smith and Michael S. Waterman, Journal of Molecular Biology, 1981, pp. 195-197.

[4] "Rapid and sensitive protein similarity searches," D. J. Lipman, W. R. Pearson, Science 22 March 1985, Vol. 227. no. 4693, pp. 1435 – 1441.

[5] Sargasso Sea project, December 2006. Available online: http://www.genomenewsnetwork.org/articles/2004/03/04/sargasso.php.

[6] "A Proposed Hybrid Approach for Patent Modeling," Ognjen Scekic, Djordje Popovic, Veljko Milutinovic, Transactions on Internet Research, July 2006, Vol. 2, Number 2.

[7] "Ontology Learning and Its Application to Automated Terminology Translation," Roberto Navigli, Paola Velardi, Aldo Gangemi, IEEE, Intelligent Systems, 2003, pp. 22-31

[8] "Transactional Memory: Architectural Support for Lock-free Data Structures," M. Herlihy, J. Eliot, B. Moss, Proceedings of the 20th Annual International Symposium on Computer Architecture, 16-19 May 1993, pp. 289-300.

[9] AMMP Home Page, December 2006. Available: http://www.cs.gsu.edu/~cscrwh/ammp/ammp.html

[10] SPEC OMP 2001 Benchmark, December 2006. Available: http://www.spec.org/omp/

[11] "A Survey of Distributed Shared Memory," Jelica Protic, Milo Tomasevic, Veljko Milutinovic, Proceedings of the 28th Annual Hawaii International Conference on System Sciences, 1995.