Applications of Parallel Computing

D.M.U.C Dassanayake, L.Y.S.G. De Silva, M.D.Y. Fernando, D.D.S.L. Jayawardana, A.I. Ranatunga
Department of Computer Science & Engineering, University of Moratuwa.

Abstract — *parallel computing is one of the main solutions for the newly arising computational problems. Main implementations of parallelism are multi-core systems. To improve the performance in specific usage such as matrix multiplication we have to analyse the algorithms behind it. Markov's chains and memetic algorithms are two algorithmic implementations of parallel programming. Data mining is used in AI development, but can be efficiently used to optimize parallel computing as well.*

Index Terms — parallel computing, multi core architecture, synchronization, data mining

I. Introduction

Parallel computing can be considered as the main solution for the arising computational problems such as time and space complexity. Even though the high level concept of parallelism is quite simple its implementations are very expensive. The main drawback of the existing parallel algorithms is, even though they are optimized in certain scenarios, there are some scenarios in which they perform abysmally. This paper analyses various parallel architectures and algorithms, and gives an insight of finding an optimal algorithm for each application based on data mining. The paper discusses on the applications of matrix multiplication in great detail.

Even though parallelism seems to be a brilliant idea, it has the problem of diminishing the return when the number of processors is increased. The journal "The Landscape of Parallel Computing Research, try to figure out the reasons for this problem by analysing parallelism from different angles. The earlier mentioned journal tries to figure out some good aspects of parallelism by finding answers to seven core questions in parallelism. They discuss about these seven critical questions throughout the report. Those questions are, [1]

1. What are the applications?
2. What are common kernels of the applications?
3. What are the hardware building blocks?
4. How to connect them?
5. How to describe applications and kernels?
6. How to program the hardware.
7. How to measure success?

II. Multi core Systems

With the high demand for speedy execution of instructions in processing units, the passion for multi core architectures has grown rapidly. Multi core architectures have been one of the best and widespread solutions for the implementation of parallel processing.

Multi core systems are now more popular even in embedded systems since they have many advantages over single core processors. Even though this trend is a big achievement in computer history, it poses several challenges. The principal challenge is parallel programming with synchronization. The above approach says "Amdahl's law limits the efficiency of parallelism for single tasks; and synchronization issues can further impede performance of a given application." Their report states that although parallel programming can be used to achieve great efficiency with multi-tasking, there is a performance degrade when it comes to single tasking.

As a solution, they suggest to generalize the old school multi-scalar processor approach into a multi core one. In this case, cores will be considered as ordinary execution units. Weak synchronization is the concept they have used in order to achieve their targets, for example allocations are done early and synchronization will be performed later. The authors of this journal have suggested a simple architecture which will support the targets mentioned above.

In that architecture, there are three major components namely, Super Scalar Processor (SSP), Allocation and Control Unit (ACU) and Advanced Processing units (APU). The general idea of the execution is to execute the "control intensive" parts of a program which are hard to parallelize, on the SSP and the compute-intensive parts on the APUs.

Even though this architecture seems to be pretty good there are some identified issues: execution time and latencies for micro-programs are expected to be much larger than for ordinary instructions, therefore allocation and synchronization times can have a huge impact. They have suggested a solution to overcome the impact of these issues. That is to co-design some specific parts of the Operating System (OS) on the control core with automatic allocation and synchronization mechanisms of the ACU. Since the OS itself considers about automatic allocation, the program does not need to worry about it. It is one of the main advantages. The heart of this design is the SSP. If a program wants to enter the main program that is done through SSP. Weak synchronization is supported by the execution model of the

design. The ACU is a standard execution unit in the SSP. It manages allocation and synchronization of processing cores.

According to a test done by Raphael David and Alexandre Guerre from the LCE laboratory in CEA, LIST [2], the formalism is achievable, and high performance can be achieved once this design is optimized. Therefore it is evident that this design well matches with the concept of weak synchronization to attain the goals of multi core architecture with parallelism.

Above we have discussed the architecture of the OS to support parallelism. It is worth to consider another approach of parallelism which is useful to optimize the efficiency of parallel programming. It is shown by experiments how the number of threads in a processor affects the processor speed in parallel programming. [3]

Space parallelization or domain decomposition is one of the most frequently used methods for high performance computing in multi-core processors. Since these approaches are not adequate, Nikita Raba, Elena Stankova and Natalya Ampilova have tried to find some other factors which will affect the speed of a processor which is in a multi-core environment. They have done an experiment to get their results with the help of software which realizes 1.5-D numerical model of a natural convective cloud. [3]

"The best results (the smallest calculation time) is obtained when the same number of threads is used for parallelizing microphysical and dynamical processes, and this number is equal to the number of processor cores" [3]. Their test experiments have shown that the maximum speed up value can be obtained when the number of cores is equal to the number of threads. As shown in above research [3], when it is further increased, that will result in decreasing the speed of the processor because of the growth of overheads for thread creation. Therefore it is important to have the number of threads in a processor at a precise value in order to achieve the required efficiency.

III. Implementation

Parallel programming, as mentioned earlier, is being put into practice due to its ability to solve problems in speed, resource allocation by concurrency. When it comes to implementation, it is really interesting to examine the use of programming languages for parallel processing and various applications that use parallel programming.

Addressing the issue of achieving flexibility in coding and developing new software to gain the real advantage of parallel computing has been a challenge over the past era. Computer languages which are able to use parallel processing play a main role in solving the issue. Whether to build a new language which has all the parallel programming aspects or to extend an existing language like C or C++ poses a question to computer scientists and engineers. Even though the former approach seems viable, programmers will not find it easy to throw away existing languages, especially C in the case of programming in between software and hardware platforms. [5]

As an implementation of parallel processing, a parallel version of the Lagrangian particle model called LAMBDA can be introduced. It uses the Message Passing Interface (MPI) library. This is an effective and reliable model to simulate the airborne pollutant dispersion. Parallel programming can be applied in the performance tests of this model.

While the Eulerian model uses a fixed reference system w.r.t. to the earth, the reference system of the Lagrangian model changes with the instantaneous changes in the atmosphere. The Lagrangian models perform well in complex environments and also with pollutant dispersion sources of arbitrary shape and size.

It can be shown that between the following two strategies which arise from the LAMBDA code, the latter is better; domain decomposition and dividing particles among processors. [5]

To show that the set of particles released are assigned to different processors to evolve. The fact that the gain in the processing time is limited to the percentage of the code that can be executed in parallel must be emphasized here. The code uses a distributed memory parallel machine:

"The related code was parallelized using calls to the message passing communication library MPI and executed on a distributed memory parallel machine. This machine is a cluster of 17 standard IA-32 architecture processors connected by a standard Fast Ethernet interconnection network and a 24-port switch."[5]

The results of the performance tests show that the decrease in efficiency with the number of parallel processors decreased is not due to the sequential part of the respective parallel code. The issue arises along with the reduction and broadcasting overhead in communication.

The graphical representation of the results of the experiment showing Final dispersion for 2000 particles generated in each one of 500 time steps for different number of processors (including one) is figured out for comparison.

IV. Algorithms

A. Memetic Algorithms

"Memetic algorithms" are used to determine the optimum number of processors and the optimum data distribution among those processors for different programs or sections of a program. Memetic algorithms are derived from the genetic algorithms by pouring in hill climbing techniques.

A research paper titled "Memetic algorithms for parallel code optimization" by Özcan, Ender and Onbaşioğlu, Esin describes about memetic algorithms and mutimememetic algorithms. [6]

The section includes a set of figures to describe the data alignment patterns and communication structures. In genetic algorithms, a population of individuals is generated

randomly. A fitness function measures the ability to survive of an individual i.e. the quality of a solution. Memetic algorithms are obtained by applying hill climbing techniques to genetic algorithms. Those algorithms contain both genetic and memetic algorithms.

Further, the paper describes the way to solve the complex problem of obtaining the optimum number of processors and best alignment for a given program. Determining the optimal number of processors given a parallel code, the number of levels in it, the different alignments and the degree of parallelism within each level is a challenging task. Different memetic algorithms are compared for PCO (Parallel Code Optimization). In multi-meme memetic algorithms, there are several strategies which can be used to select the meme, for instance, the meme of the best individual in the population.

Related to research mentioned in the above paper, experiments have been performed using data sets along with twelve problem instances. The Hessenberg reduction was used in the first and second data sets and the Dongarra's benchmark was used for the fourth data set. More new experimental data sets can be generated by mixing the above mentioned data sets.

Finally, it describes the experimental results. Common experimental settings detail out the machine. Success rate denotes the probability that a user finds a solution in a single memetic algorithm for a given problem instance. [6]
These kinds of algorithms will give an alternative to use other than time consuming brute force. Using suitable heuristics will further improve the performance.

B. Markov's Chain Algorithms

The research paper "Parallel algorithms for simulating continuous time markov chains" by David nicol and philipheidelburger describes a mathematical way of defining the dependencies between processes. It further states that logical processes should be assigned to processors very carefully. [12] According to the paper, continuous time markov's chain includes a function x(t) as the state of CTMC. This state is simply a vector that indicates the queue lengths of the networks. It also defines a term called "holding time" which stands for the time within the state transition. This "holding time" follows an exponential distribution.

Then the process which has the least transition time is evaluated according to the exponential probability distribution. That defines the whole vector's transition time. For the ease of simulation the possible transitions are divided in to 2 sets. One is the transitions which does not affect the logical processes. Other set is the one that affect some logical processes.

Sampling the holding time into discrete values is also essential because computers can process only discrete time data. At last "Uniformization" is done according to a common parameter λmax which is greater than all the exponential distribution parameters of other distributions.

It is important to have a look at some specific applications for which we can effectively apply our knowledge to overcome the main design problem as stated at the beginning of our paper. One of these interesting applications is matrix multiplication.

"A High performance Two Dimensional Scalable parallel algorithm for solving sparse triangular systems" by Mahesh V. Joshi describes an algorithm to solve linear systems which were simplified to L, U factorized level. As they propose, they introduce the first ever known efficient scalable algorithm which uses two dimensional block cyclic distribution of 'T'. Here 'T' refers to the coefficient matrix of the linear system.

Since there are really fast parallel algorithms for factorization of matrices there is a bottle neck which is possible in getting the final solution after factorization. This is why they thought of implementing a fast parallel algorithm for already factorized matrices.

Once the variable x1 which multiplies with the corresponding diagonal element is evaluated for forward substitution, all the elements that lie below it should also be updated with that value. (In a lower triangular matrix) For example (if x1 is evaluated, a21x1,a23x1.....can also be updated in parallel.) This can be parallelized because this update is not affected by anything else. [13]

Another research paper "Analysis of a class of parallel matrix multiplication algorithms" published by the University of Texas at Austin gives a clear idea about distributing computational tasks of matrix multiplication between processors. Here it suggests a computational mesh consisting of 'r' rows and 'c' columns. [8]

Main focus of this paper is to emphasize the fact that communication overheads always play a major role in matrix multiplication.

As the paper describes the main drawback of existing algorithms is that there are no algorithms which are defined for specific sizes of matrices. Size of the matrix here refers to the [8] number of rows (m), number of columns (n) and (k) which is the matching number of rows and columns of the matrices being multiplied.

There are special cases such as any one of them becomes one or all of them become one. In case if all m, n, k becomes one it would simply be a scalar multiplication.

The cost between the communication links should be calculated. Paper uses the formula [8]

c=a+nb

Where 'a' is the starting cost, 'n' is the length of the message and 'b' is the cost per byte. Minimum spanning tree of the communication graph should be created to remove unnecessary links. These models relate to communication costs.

Most interesting part is distributing blocks of the partitioned matrix among the nodes in a logical mesh. For this partition we consider two vectors 'x' and 'y', where 'x' is

partitioned into M while 'y' is partitioned into N. Size of one partition is named as "block distribution". Eventually block size of a matrix will become bd2. ("bd" stands for block distribution)

It is clear to see that above paper also raises the question about trade-off between the algorithmic efficiency and communication efficiency. Another special example is the case of sparse matrixes.

"Two dimensional data distribution method for parallel sparse matrix vector multiplication" by Brendan Vastenhouw and Rob H. Bisseling suggests that the recursive bi-partitioning of the matrix is done in a way that is would distribute communication and computational work evenly among the processors.

Algorithm is so simple. [7] Each processor should send vj (jth element of the vector) components to the processors which have aij (ijth element of the matrix A) s. They have to multiply those nonzero aijs with corresponding vjs and each processor should add the received components of ui. (ith element of the resulting vector). By combining those results we can arrive at the resulting vector.

Surprisingly, this also points to the same issue of communication costs. One metric of the communication cost is the size of all the data words sent and received between processors. With this communication optimization, redundancy should be eliminated or minimized. [7] For instance, if one processor needs vj component twice, communication should only happen once.

Distribution of the aij s would be in a one dimensional fashion or a two dimensional fashion. When it is one dimensional each row can be distributed among each processor at a moment. This will remove the phases of receiving and sending components of row sums to other processors. At the same time number of destination processors for vj reaches p where p is the total number of processors which gives an adverse effect in distribution of data related to vj. In this one dimensional distribution, number of non-zero elements may not also be uniformly distributed. However in 2D distribution number of destination processors for vj reaches $\sqrt{P-1}$

It is necessary to mention that matrix partitioning problem generalizes to a graph partitioning problem where an edge between i and j in the graph represents a non-zero aij element. Main purpose of partitioning is to minimize the cut edges. A cut edge occurs when the vertices i and j elements corresponding to a non-zero element aij are assigned to different processors. However this graph partitioning generalization can only be used for square matrices.

One conclusion we can make from the above is that we can't select the most optimum algorithm for matrix multiplication without knowing the distribution of the matrix entries.

V. Data Mining

It is a huge leap we had from uniprocessor to multiple processors. Data mining is also a vast area which can be occasionally used to optimize parallel computing.

Web Mining is the use of data mining techniques to automatically discover and extract information from Web documents and services. [9] There are four main techniques involved.

1. Resource finding: getting web resources.
2. Information selection and pre-processing: pre-process retrieved content
3. Generalization: identify patterns
4. Analysis: interpret patterns

[10] All these techniques are involved with two main processes. Information retrieval (IR) and information extraction (IE). In fact, IR is the automatic retrieval of all relevant documents and, while recovering a portion of the non-relevant as possible. IE is the transformation of a collection of documents, usually with the help of an infrared system in which the information is better digested and analysed. For dynamic contents, it is not easy to build such a system.

Data mining is more convenient with the automatic retrieval of documents. Even the research papers have become available on the internet. So there is a need of agents searching the internet, an automated means to find, download, and the judge relevance of the researches published.

Some web sites provide indexes which ease the searching process. Some are static HTML where a search engine should be used to browse data. But most of web content is postscript. There may be too many potentially interesting papers and repetitive user intervention for reading the same content.

Evolution of data mining techniques can be used in such a way to select the most appropriate algorithm to multiply two matrices with the help of a knowledge base. More knowledge on artificial intelligence will guide through our research.

VI. Conclusion

We have discussed the hardware implementation of multi core systems and the theory behind parallel computing and the algorithms that are developed to optimize parallelism. A different view point of the parallelism is data mining. More the data dependencies can be analysed; more the parallelism can be exploited. Data mining field will get expanded more and more with the development of artificial intelligence. Hence, the technologies we discussed here may be obsolete in the near future.

REFERENCES

[1] Kryste Arsanovic 1998."The Landscape of Parallel Computing Research: A View from Berkeley" University of California at Berkeley researchers Dec2006 [Online]
Available:http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf [6 Jan 2012]

[2] Stephani Louise,Vincent David, Fablen Calcado 2011. "A single processor approach for loosely synchronized execution of parallel flows on heterogeneous multi core". Procedia Computer Science, Proceeding of the international conference of the computer science, Vol 4, 2011, [Online] Available: [28 Jan 2012]
http://www.sciencedirect.com/science/article/pii/S1877050911002766

[3] Natalya Ampilova,Nikita Raba,Elena Stankova, 2010 ,"On investigation of parallelization effectiveness with the help of multi-core processors", Procedia Computer Science, Proceeding of the international conference of the computer science, Vol 1,issue 1, May2010, [Online] Available: [28 Jan 2012]
http://www.sciencedirect.com/science/article/pii/S187705091000311X

[4] Chris Edwards, 2008,"Parallel pain",E&T engineering and technology magazines Jun2008 Vol 3 issue 2 [Online] Available:http://eandt.theiet.org/magazine/2008/02/parallel-pain.cfm [28 Jan 2012]

[5] Roberti, Debora; Souto, Roberto; Velho, Haroldo; Degrazia, Gervasio; Anfossi, Domenico; " Parallel Implementation of a Lagrangian Stochastic Model for Pollutant Dispersion," International Journal of Parallel Programming, Oct2005, Vol. 33 Issue 5, p485-498, 14p; DOI: 10.1007/s10766-005-7302-z
[Online]
Available:http://web.ebscohost.com/ehost/pdfviewer/pdfviewer?sid=f3f5b934-12c4-413c-990c-cdf0534233a7%40sessionmgr12&vid=1&hid=21 [28 Jan 2012]

[6] Özcan, Ender; Onbaşıoğlu, Esin; " Memetic Algorithms for Parallel Code Optimization," International Journal of Parallel Programming, Feb2007, Vol. 35 Issue 1, p33-61, 29p, 6 Diagrams, 13 Charts, 2 Graphs; DOI: 10.1007/s10766-006-0026-x
[Online]
Available:http://web.ebscohost.com/ehost/pdfviewer/pdfviewer?sid=4fb28492-00a4-42d4-96e1-bf8603e17bdf%40sessionmgr11&vid=1&hid=21 [28 Jan 2012]

[7] Brendan Vastenhouw, Rob H. Bisseling. 2005 "Two dimensional data distribution method for parallel sparse matrix --vector multiplication" SIAM REVIEW Vol. 47,No. 1,pp. 67–95 [online] available: http://igitur-archive.library.uu.nl/math/2011-0331-200620/4-40901.pdf [29 Jan 2012]

[8] Gunnels, J.; Lin, C.; Morrow, G.; van de Geijn, R.; "A flexible class of parallel matrix multiplication algorithms," Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International ... and Symposium on Parallel and Distributed Processing 1998 , vol., no., pp.110-116,30doi:10.1109/IPPS.1998.669898:[online]
available:http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=669898&isnumber=14764[29 Jan 2012]

[9] Raymond Kosała, Department of Computer Science, KatholiekeUniversitei Leuven Celestijnenlaan 200A, B-3001 Heerlen, Belgium
Raymond@cs.kuleuven.ac.be
Vol no: 2, pp.1-5
July 2000 [Online] Available: http://arxiv.org/pdf/cs.LG/0011033.pdf[5 Jan 2012]

[10] Katia P. Sycara ,Michael Wooldridge,1998 ,An Autonomous Web Agent for Automatic Retrieval and Identification,of Interesting Publications, Proceedings of the 2nd International Conference on Autonomous Agents,ACM Press, New York, pp. 116–123, 1998. vol no [Online]Available:http://comet.lehman.cuny.edu/jung/html/teaching/cmp788DIR/citeseer1.pdf [7 Jan 2012]

[11] 2010. "Mix and match for multicore" E &T engineering and technology magazine Vol 5 issue 5 Mar2010 [Online] Available :http://eandt.theiet.org/magazine/2010/05/multicore-mashup.cfm

[12] David M. Nicol, Philip Heidelberger 1993."Parallel algorithms for simulating continuous time Markov chains" Proceeding PADS '93 Proceedings of the seventh workshop on Parallel and distributed simulation ACM New York, NY, USA[Online]Available:http://delivery.acm.org/10.1145/160000/158461/p11-nicol.pdf?ip=192.248.8.106&acc=ACTIVE%20SERVICE&CFID=75844679&CFTOKEN=59370427&__acm__=1325243043_ef540cfd6aaa0ffdec982650e2eee131

[13] Mahesh V. Joshi, Anshul Gupta, George Karypis, Vipin Kumar, 1997. "A High Performance Two Dimensional Scalable Parallel Algorithm for Solving Sparse Triangular Systems ", 4th International Conference on High Performance Computing, (HiPC'97) [Online] Available: http://www-users.cs.umn.edu/~kumar/papers/papers.html[6 Jan 2012]