Approaches to Data Parallel Programming

Vishaldeep Singh, Arun Seth

Assistant Professor, Dept. Of Computer Science & Engineering, Guru Nanak Dev University Amritsar, Punjab, India
Assistant Professor, Dept. Of Computer Science & Engineering, Guru Nanak Dev University Regional Campus Gurdaspur, Punjab, India

---***---

Abstract - *In this paper efforts have been put efforts to illustrate in the best way how to start with the programming of data parallel model. The various things that need to be taken care before starting up with this approach of programming.*
Data Parallel Approach means split the data on which the instruction is to be applied and assign the same task to different processing elements for the processing over the individual data that has been assigned to them. Hence after all the processing elements are done with the assigned task then the whole result is then accommodated back at one place.

Two approaches of data parallel programming have been discussed: (i) Using multithreading in Java (ii) Using DPCE i.e. Data Parallel C Environment. The first approach although uses all the available cores of the CPU to optimal use but gives the programmer very less control over the data distribution. But the second approach gives more ability to programmer to change the data and decide how the data is to be distributed among the various available processors and various cores in these processors.
Then finally an optimal approach using genetic algorithm is discussed. It discusses how to optimize the data parallel distribution among the various processors. Then the challenges faced during programming are discussed.

Key Words: **Javaspace, Genetic Algorithm, Cross-over, Mutation, Java Multi-threading, Traversing Salesman Problem.**

1. INTRODUCTION

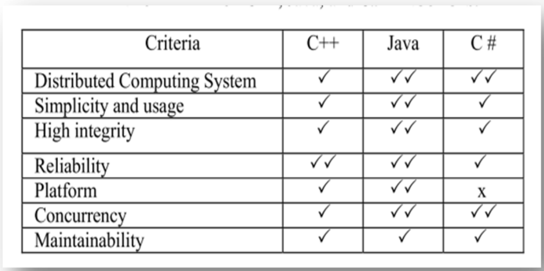
Need for programming languages in parallel processing environment:

- Current languages do not provide support for distributed applications
- Heterogeneous operating system on various clients and servers
- Data is not local and is distributed over the whole network
- Absence of concurrency amongst data

Problems:

- Developers need to perform a lot of work.
- They must have very good knowledge of these languages.
- Interoperation is still not very efficient.

With the advances in hardware and communication technology, concurrency and distribution have come naturally into a large spectrum of applications, as they try to cope with a world where people and information are geographically distributed and where several things can happen simultaneously. Concurrency and distribution introduce a set of problems that greatly increase the complexity of the software. First, and most importantly, concurrent and distributed systems are inherently more complex than non-distributed systems. Secondly, **existing programming models** and languages that are appropriate for sequential, centralized systems don't necessarily provide the appropriate mechanisms for effectively expressing concurrent and distributed scenarios.



Criteria	C++	Java	C #
Distributed Computing System	✓	✓✓	✓✓
Simplicity and usage	✓	✓✓	✓
High integrity	✓	✓✓	✓
Reliability	✓✓	✓✓	✓
Platform	✓	✓✓	x
Concurrency	✓	✓✓	✓✓
Maintainability	✓	✓	✓

Table 1: Comparison amongst various languages in distributed system [4]

2. METHODOLOGY

<u>Data Sharing</u>
Localized sharing can improve memory bandwidth efficiency

- Efficient memory bandwidth usage can be achieved by synchronizing the execution of task groups and coordinating their usage of memory data

Efficient use of on-chip, shared storage

- Read-only sharing can usually be done at much higher efficiency than read-write sharing, which often requires synchronization

<u>Data Sharing Example - Matrix Multiplication</u>
Each task group will finish usage of each sub-block of N and M before moving on

N and M sub-blocks loaded into Shared Memory for use by all threads of a P sub-block
Amount of on-chip Shared Memory strictly limits the number of threads working on a P sub-block

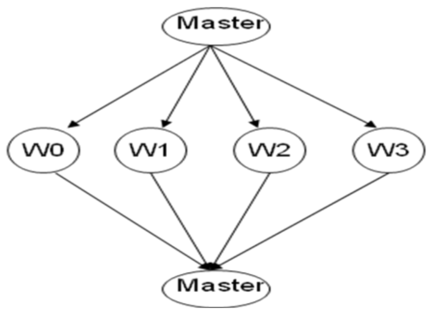


Fig 1: Master-slave model

Master:

- Creates workers
- Sends initial values to workers
- Receives local "sum"s from workers
- Calculates and prints "pi"

Workers:

- Receive initial values from master
- Calculate local "sum"s
- Send local "sum"s to Master

3. JAVA MULTI-THREAD ENVIRONMENT

Java Spaces

The notation followed by [4] is used here and is elaborate and explained in details as follows:

JavaSpace model is a high level coordination toll for connecting processes together in a distributed environment. [4] The technology provides a different programming model, which *views an application as a collection of processes* cooperating via flow of copied objects into or out of one or more spaces.

A space is a shared, network accessible repository for objects. [4] Processes use the repository as persistent object storage. Processes perform a simple operation to write new objects into space, take object from space or read object in space. For taking an objects, processes uses a **value matching lookup** to find the required object. If a matching object is found immediately then a process can perform the required operation, otherwise process has to wait. **Processes do not modify object in space**

Fig 2: Java Space Model [4]

The notation followed by [2] is used here and is elaborate and explained in details as follows :

The JavaSpace Model:

JavaSpace model is a high level coordination toll for connecting processes together in a distributed environment. [2] The technology provides a different programming model, which *views an application as a collection of processes* cooperating via flow of copied objects into or out of one or more spaces. [2] A space is a shared, network accessible repository for objects. Processes use the repository as persistent object storage [2]. Processes perform a simple operation to write new objects into space, take object from space or read object in space. For taking an objects, processes uses a **value matching lookup** to find the required object. If a matching object is found immediately then a process can perform the required operation, otherwise process has to wait. **Processes do not modify object in space.** To modify an object, a process must explicitly remove it, update and reinsert into the space. During the updating of the object, other processes have to wait. [2]

Features of JavaSpace [2]:

- **Spaces are persistent:** Spaces provide reliable storage for objects. Once stored in the space, an object will remain there until a process explicitly remotes it.
- **Spaces are persistent:** Spaces provide reliable storage for objects. Once stored in the space, an object will remain there until a process explicitly remotes it.
- **Spaces are persistent:** Spaces provide reliable storage for objects. Once stored in the space, an object will remain there until a process explicitly remotes it.
- **Spaces are transitionally secure:** The Space technology provides a transaction model that ensures that an operation on a space is atomic. Transactions are supported for single operations on

a single space, as well as multiple operations over one or more spaces.

- **Spaces are transitionally secure:** The Space technology provides a transaction model that ensures that an operation on a space is atomic. Transactions are supported for single operations on a single space, as well as multiple operations over one or more spaces.

- **Spaces are transitionally secure:** The Space technology provides a transaction model that ensures that an operation on a space is atomic. Transactions are supported for single operations on single space, as well as multiple operations over one or more spaces.

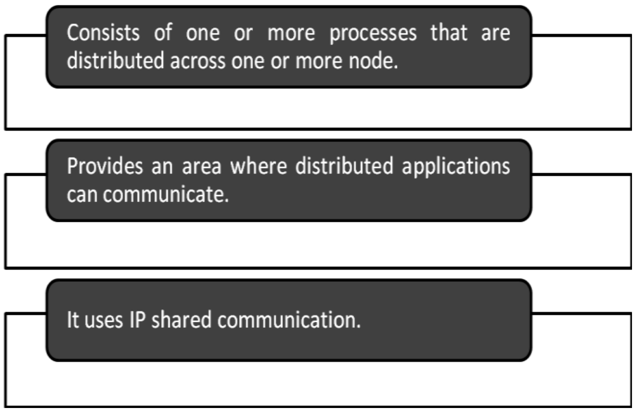

Consists of one or more processes that are distributed across one or more node.
Provides an area where distributed applications can communicate.
It uses IP shared communication.

Fig 2: Javaspace implementation[2]

4. DATA PARALLEL C ENVIRONMENT

This gives the programmer a freedom to control the assignment of data to the various nodes unlike the java threading counterpart of the same. Here the programmer can decide that the data which is to be assigned to which

Problems solved in DPCE [1]:

- It allows processes on different machines to share data.

- It allows using user defined data-types on various nodes of the distributed system.

- This may be achieved my replicating the data on the various nodes of the distributed system.

- Even if there is no physical shared memory even then the processes here can communicate with each other in DPCE.

5. GENETIC ALGORITHM FOR OPTIMIZED DATA PARALLEL PROGRMMING

Optimizing By Genetic Algorithm [5]

The basic form of the GA. This algorithm will be applied to each of the randomly generated queries used in an experiment.

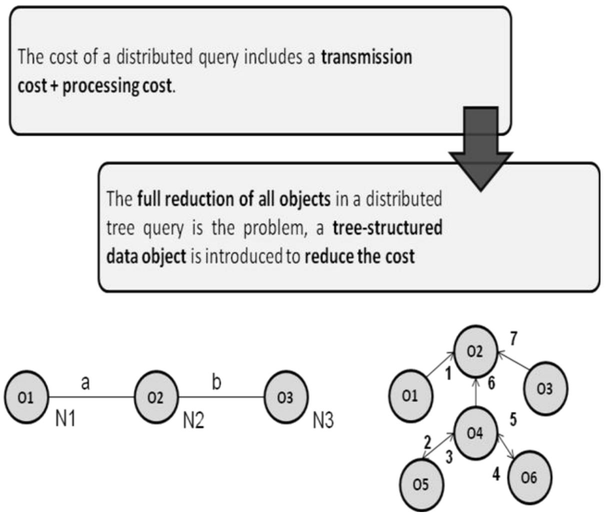

The cost of a distributed query includes a transmission cost + processing cost.
The full reduction of all objects in a distributed tree query is the problem, a tree-structured data object is introduced to reduce the cost
01
a
02
b
03
N1
N2
N3
01
02
03
04
05
06
1
2
3
4
5
6
7

Fig 3: Node graph signifies all the nodes [5]

5.1 Initialization Phase

In this phase of the algorithm an initial population of random solutions is generated for the target query. Generating a random solution includes randomly selecting a root table and the type (single or double) of each edge. The choice of root table is made in sympathy with the number of root-table-reducing schedules for each candidate root table.

5.2 Selection Phase

In this phase of the: algorithm, solutions are selected to participate in the reproduction phase of the algorithm based on relative fitness. A solution is a root-table-reducing semi-join schedule. The fitness function is given by: schedule cost + cost to fully reduce non-root tables [5]

5.3 Reproduction Phase

In this phase of the algorithm, new solutions (schedules) are generated using a crossover operator. The crossover operator works by exchanging a sub-schedule between a pair of compatible schedules. The process starts by selecting the first schedule to be crossed. From the join tree of the selected schedule, a sub-tree is chosen for the exchange. The schedule for this sub-tree is then exchanged with the schedule (for the same sub-tree) used by a randomly-selected, compatible crossing partner (a schedule where the chosen sub-tree root has the same parent table).

Cross-Over Operation [5]

Join trees (a) and (b) in Figure depict two schedules that are compatible if the chosen sub-tree root is T4; they are incompatible if the chosen sub-tree root is T2.

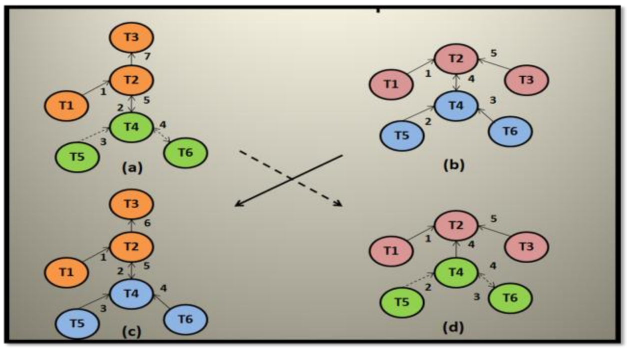


Fig. 4: To illustrate the crossover operation in GA [5]

Mutation [5]

During mutation, one of five mutation operators is applied to one randomly selected table:

(a) change a single edge to a double edge,

(b) change a double edge to single edge,

(c) change order of double edges,

(d) (non-root only) set table as new root, and

(e) (root only) change last edge

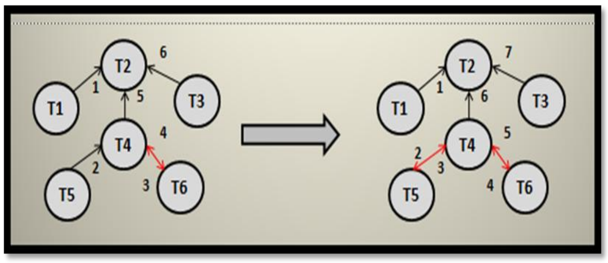


Fig. 5: Change a single edge to a double edge [5]

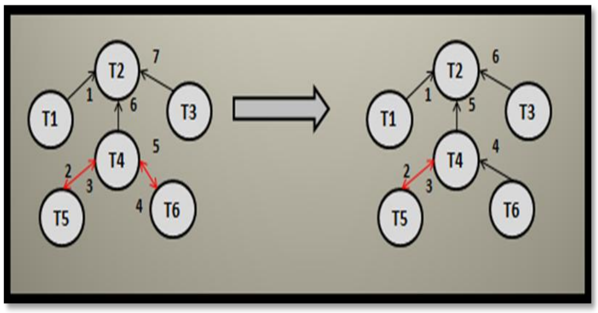


Fig 6: Change a double edge to single edge [5]

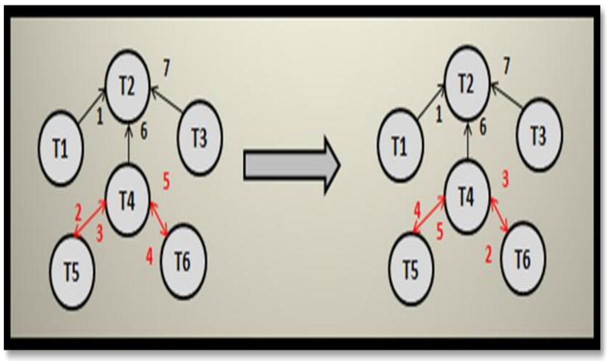


Fig. 7: Change order of double edges [5]

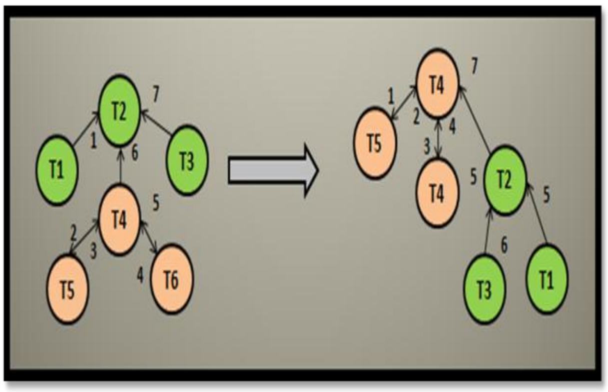


Fig 8: (non-root only) set table as new root [5]

Full Implementation

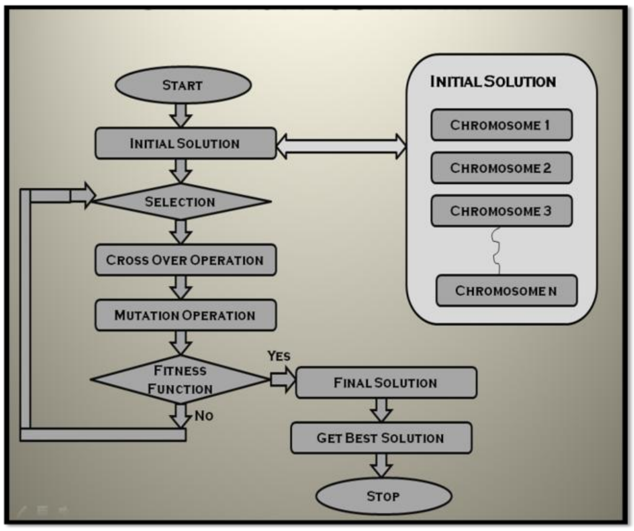


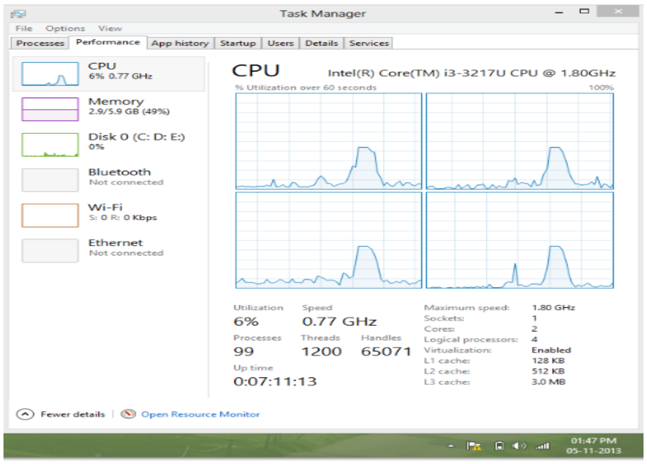
Fig. 9: Full GA Implementation

6. IMPLEMENTATION AND SELF ANALYSIS

(i) Case Study 1. (Using Java Multithreaded programming)

The java program that is considered here [6] makes almost 500 threads and then makes them execute on different cores of the processor because by default the processor will make the cores work equally in proportions.

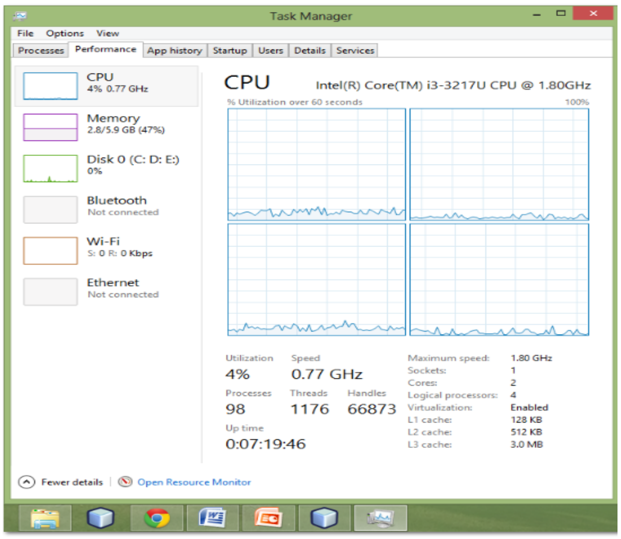
The CPU utilization for the parallel program is:



```
Utilization   Speed                     Maximum speed:       1.80 GHz
6%            0.77 GHz                  Sockets:             1
Processes     Threads     Handles       Cores:               2
99            1200        65071         Logical processors:  4
Up time                                 Virtualization:      Enabled
0:07:11:13                              L1 cache:            128 KB
                                        L2 cache:            512 KB
                                        L3 cache:            3.0 MB
```

Fig 10: The CPU utilization running 500 threads

The normal Execution of CPU under all the same programs is as shown except for running the program is as shown below:

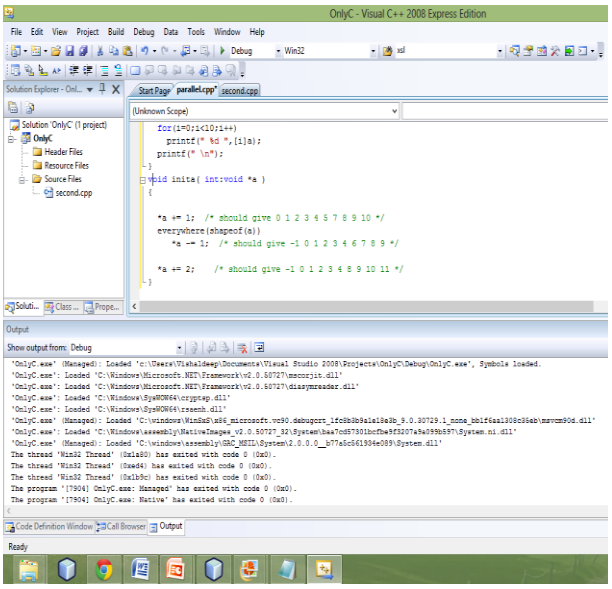


```
Utilization   Speed                     Maximum speed:       1.80 GHz
4%            0.77 GHz                  Sockets:             1
Processes     Threads     Handles       Cores:               2
98            1176        66873         Logical processors:  4
Up time                                 Virtualization:      Enabled
0:07:19:46                              L1 cache:            128 KB
                                        L2 cache:            512 KB
                                        L3 cache:            3.0 MB
```

Fig 11: The CPU utilization when idle

(ii) Case Study 2. (Using Data Parallel C Environment, DPCE)

The execution of the data parallel C program which can control the data that is granted to each of the cores of the processors is as below [6]:

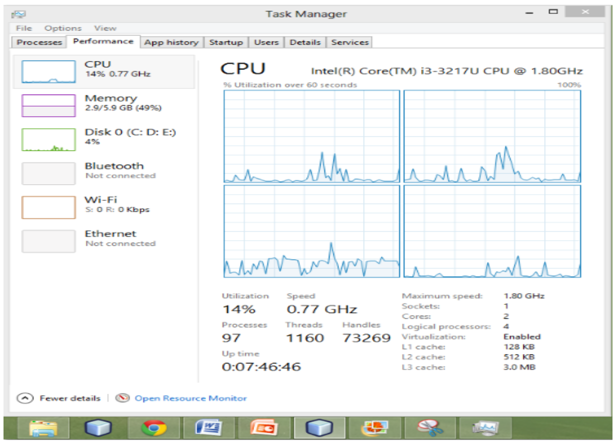


```
    for(i=0;i<10;i++)
      printf(" %d ",[i]a);
    printf(" \n");
}
void inita( int:void *a )
{


    *a += 1;  /* should give 0 1 2 3 4 5 7 8 9 10 */
    everywhere(shapeof(a))
       *a -= 1;  /* should give -1 0 1 2 3 4 6 7 8 9 */

    *a += 2;    /* should give -1 0 1 2 3 4 8 9 10 11 */
}
```

```
'OnlyC.exe' (Managed): Loaded 'c:\Users\Vishaldeep\Documents\Visual Studio 2008\Projects\OnlyC\Debug\OnlyC.exe', Symbols loaded.
'OnlyC.exe': Loaded 'C:\Windows\Microsoft.NET\Framework\v2.0.50727\mscorjit.dll'
'OnlyC.exe': Loaded 'C:\Windows\Microsoft.NET\Framework\v2.0.50727\diasymreader.dll'
'OnlyC.exe': Loaded 'C:\Windows\SysWOW64\cryptsp.dll'
'OnlyC.exe': Loaded 'C:\Windows\SysWOW64\rsaenh.dll'
'OnlyC.exe' (Managed): Loaded 'C:\windows\WinSxS\x86_microsoft.vc90.debugcrt_1fc8b3b9a1e18e3b_9.0.30729.1_none_bb1f6aa1308c35eb\msvcm90d.dll'
'OnlyC.exe': Loaded 'C:\Windows\assembly\NativeImages_v2.0.50727_32\System\baa7cd57301bcfbe9f3207a9a099b597\System.ni.dll'
'OnlyC.exe' (Managed): Loaded 'C:\windows\assembly\GAC_MSIL\System\2.0.0.0__b77a5c561934e089\System.dll'
The thread 'Win32 Thread' (0x1a80) has exited with code 0 (0x0).
The thread 'Win32 Thread' (0xed4) has exited with code 0 (0x0).
The thread 'Win32 Thread' (0x1b9c) has exited with code 0 (0x0).
The program '[7904] OnlyC.exe: Managed' has exited with code 0 (0x0).
The program '[7904] OnlyC.exe: Native' has exited with code 0 (0x0).
```

Fig 12: Snapshot of visual C++ executing code

The CPU utilization of different cores if different as the data given to them for processing is uneven:



CPU 14% 0.77 GHz
Memory 2.9/5.9 GB (49%)
Disk 0 (C: D: E:) 4%
Bluetooth Not connected
Wi-Fi S: 0 R: 0 Kbps
Ethernet Not connected

CPU Intel(R) Core(TM) i3-3217U CPU @ 1.80GHz

Utilization 14%
Speed 0.77 GHz
Processes 97
Threads 1160
Handles 73269
Up time 0:07:46:46

Maximum speed:	1.80 GHz
Sockets:	1
Cores:	2
Logical processors:	4
Virtualization:	Enabled
L1 cache:	128 KB
L2 cache:	512 KB
L3 cache:	3.0 MB

Fig. 13: CPU utilization running the parallel code

7. PERFORMANCE ANALYSIS

The notation followed by [1,2,3] is used here and is elaborate and explained in details as follows :

<u>1. Travelling Salesman Problem [1]</u>

The program uses two shared objects: a job queue and an IntObject containing the length of the current best path. [1] It should be clear that reading of the current best path length will be done very often, but since this is a local operation, there is no communication overhead. Updating the best path happens much less often, but still only requires one broadcast message.

The performance of the traveling salesman program (for a randomly generated graph with 12 cities) is given in Figure 10. The implementation achieves a speedup close to linear. With 16 CPUs it is 14.44 times faster than with 1 CPU

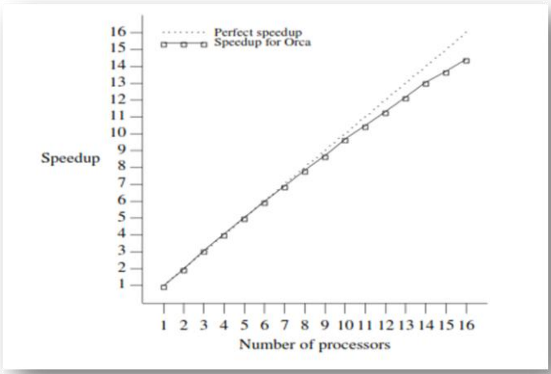


Fig. 15: Measured speedup for the DPCE implementation of the Traveling Salesman Problem. [1]

<u>2. Parallel All pair shortest path problem [2]</u>

The performance of the program (for a graph with 300 nodes) is given in Figure 4.2. The parallel algorithm performs 300 iterations; after each iteration, an array of 300 integers is sent from one processor to all other processors. In spite of this high communication overhead, the implementation still has a good performance. With 16 CPUs, it achieves a speedup of 15.88. One of the main reasons for this good performance is the use of broadcast messages for transferring the array to all processors

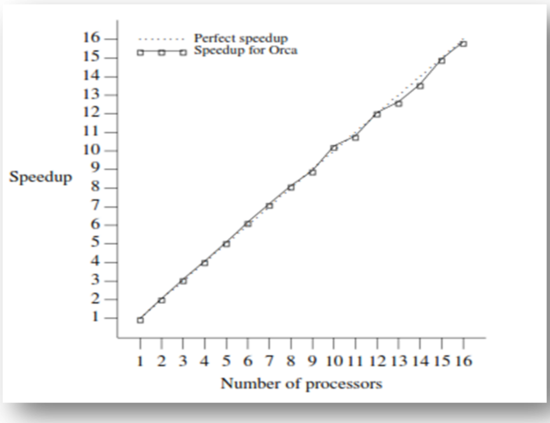


Fig. 16. Measured speedup for the DPCE implementation of the All-pairs Shortest Paths problem. [2]

8. CHALLENGES IN DATA PARALLEL PROGRAMMING

Finding and exploiting concurrency often requires looking at the problem from a non-obvious angle

- Computational thinking
- Dependences need to be identified and managed
- The order of task execution may change the answers
- Obvious: One step feeds result to the next steps
- Subtle: numeric accuracy may be affected by ordering steps that are logically parallel with each other

- Performance can be drastically reduced by many factors
- Overhead of parallel processing
- Load imbalance among processor elements
- Inefficient data sharing patterns

Saturation of critical resources such as memory bandwidth.

9. CONCLUSION

Two approaches of data parallel programming have been discussed: (i) Using multithreading in Java (ii) Using DPCE i.e. Data Parallel C Environment. The first approach although uses all the available cores of the CPU to optimal use but gives the programmer very less control over the data distribution. But the second approach gives more ability to programmer to change the data and decide how the data is to be distributed among the various available processors and various cores in these processors.

10. FUTURE WORK

With the increase in demand of the parallel programming these days there are not many experienced programmers available for doing these kinds of jobs.

The sole reason behind this is the lack of support of the traditional languages like Java and C++ to the parallel programming. Of course there are some specialised languages for parallel programming such as Drayd and DraydLINQ and some .NET platform oriented platform by Microsoft which specialises in dealing with parallel programming problems.

But programmers need time to learn and master such new ideas and concepts. Hence work can be done to implement new libraries and more support in the traditional languages to provide good exposure to the parallel programming. Like the dpce.h is given for the data parallel C environment header file which is freely available on the internet. But no support is provided to such content. Hence further some good resources can be provided for the new users to learn these concepts.

REFERENCES

1. Petr Pospichal, Jiri Jaros, and Josef Schwarz , " Parallel Genetic Algorithm ",IEEE, 2010
2. H.E. Bal, R. van Renesse, and A.S. Tanenbaum, “Implementing Distributed Algorithms Using Remote Procedure Calls,” *Proc. AFIPS Nat. Computer Conference, 2006*
3. Cristina Isabel, Videira Lopes. "DPCE: A Language Framework For Distributed Programming", IEEE, 2012
4. Michael Isard, Yuan Yu. "Distributed Data-Parallel Computing Using A High-Level Programming Language", IEEE, 2011
5. Rachid Guerraoui, Luís Rodrigues. "Introduction to Reliable Distributed Programming", Springer, 2008
6. Internet Resource:

   (i) Data Parallel C Tutorial,

   http://www.crescentbaysoftware.com/dpce/tutorial.html

   (ii) Java Data Concurrency Tutorial

   http://www.vogella.com/articles/JavaConcurrency/article.html