Computer Engineering and Intelligent Systems
ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online)
Vol.4, No.13, 2013

Parallel Query Processing on 2D Mesh and Linear Array Architectures

Amal Elsayed Aboutabl
Computer Science Department, Faculty of Computers and Information,
Helwan University, Ain Helwan, Cairo, Egypt E-mail: aaboutabl@helwan.edu.eg

Abstract
As the size of the web grows, it is necessary to parallelize the process of retrieving information from the web. Incorporating parallelism in search engines is one of the approaches towards achieving this aim. This paper presents an algorithm for query processing on the 2D mesh architecture and two algorithms for linear array architectures. We attempt to exploit the arrangement of processors and the communication pattern in both 2D mesh and linear array architectures to attain high speedup and efficiency for queries-keywords comparisons. A cost model is presented for each algorithm based on both processing and communication cost. Proposed algorithms are evaluated using speedup and efficiency performance metrics. For the same number of processors, 2D Mesh_QP outperforms both linear array algorithms (LA_QPAKP and LA_QPKE).
Keywords: 2D Mesh, Linear Arrays, Parallel computing, Query processing

1. Introduction

In addition to the increase in the size of the web and the number of available documents, there has also been an increase in the number of internet users. This has, in turn, led to a dramatic increase in the number of queries to be processed (Cho & Garcia-Molina 2002). Parallel query processing is needed to design web search engines to deal with both query traffic and the huge amount of information available on the web (Marin *et al.* 2010). Every document available on the web is associated with a number of keywords which may be words appearing in or topics covered by the document (Baeza-Yales & Ribeiro-Neto 1999). For a search engine to process a query, it needs to compare the keywords appearing in the query with the available indexed keywords in order to retrieve related documents. This work focuses on this comparison step which we attempt to parallelize for a large number of queries and keywords. In particular, algorithms are proposed here for query processing on the 2D mesh and linear array parallel architectural models.

A two-dimensional mesh is a network that can be represented in a manner as shown in figure 1(a). It can be viewed as an N×N array of processing elements (PEs). Every PE is indexed by a 2-tuple (i,j) where $0 \leq i \leq N$ is the row index and $0 \leq j \leq N$ is the column index (Perhami 2002). Such a mesh has N^2 PEs and typically adopts the local memory model where each PE has a processor and a local memory connected to it. Data are transferred from one processor to the other by routing messages through the mesh. The torus architecture is one variant of the 2D mesh where PEs on the sides are connected to those on the other side. Every PE (i,j) is connected to (i, (j+1) mod N), (i, (j-1) mod N), ((i-1) mod N, j) and ((i+1) mod N, j). Hypercubes and 3D meshes are also other variations on meshes. In 2D meshes, propagation delay between adjacent processors is quite small which facilitates high speed communication due to the short local connections between processors. Links between processors are bidirectional and capable of carrying data in both directions concurrently. The mesh can be indexed in a row-major or column-major order. A linear array can be considered as a 1D mesh as processing elements are connected in a chain-like manner. Each processor communicates with its two neighbors directly as in figure 1 (b). In a bidirectional linear array, input can be fed in from the two ends of the chain where the inputs are propagated in one direction (El-Rewini & Abd-El-Barr 2005).

In this paper, we attempt to exploit the arrangement of processors and the communication pattern in both 2D mesh and linear array architectures to attain high speedup and efficiency for queries-keywords comparisons.

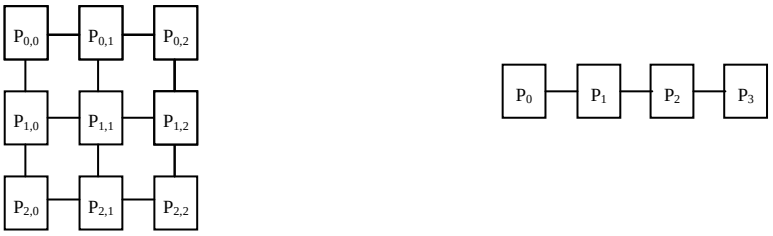
2. Related Work

Some parallel algorithms are based on the idea of domain decomposition in which a certain domain of interest is partitioned prior to computation. This is typically applied on numerical problems where the domain of interest is a matrix, vector or geometries (Panitanaraka & Shontza 2011). The divide-and-conquer approach has been one of the most common approaches in parallel algorithm design. It relies on splitting the original problem into a number of

sub-problems that can be solved in parallel. Sub-problem solutions are then merged to form the whole solution (Blelloch & Maggs 1996). Classically, parallel architectures attracted the attention of researchers to design parallel algorithms for different types of problems. Typical examples of such algorithms are sorting, searching and graph problems (Grama *et al.* 2003). Moreover, many compute-intensive algorithms such as weather forecasting and simulations require parallel processing. Data-intensive applications may also benefit from parallel processing if data-partitioning schemes are adopted.

The 2D mesh has been used to solve a variety of problems. Some classic algorithms are available in the literature to exploit parallelism on 2D meshes for searching, sorting and matrix operations such as transpose and multiplication. A bitonic sorting algorithm has been designed for 2D meshes (Ceterchi *et al.* 2007). Mesh architecture was used to solve the maze routing problem (Ercal & Lee 1997). Choi and Park (2012) parallelized a video compression algorithm on a multicore system arranged as a 2D mesh topology using a wavefront scheme to break dependencies among partitioned code blocks. An algorithm for complete exchange on 2D mesh multiprocessors is also presented (Young-Joo & Yalamanchili 2000).

A number of parallel query processing approaches have been attempted. One of the first attempts was the study of parallel query processing on shared-everything parallel systems (Hong & Stonebraker, 1991). Konstantopoulos, Mamalis, Pantziou and GavalasHong (2009) proposed parallel algorithms for document retrieval on Bulk Synchronous Processors and Coarse-Grained Multiprocessors. A study based on distributed indexing where the document index is partitioned among distributed cluster servers was also presented (Marin *et al.* 2010). Another approach (Büttcher *et al.* 2010) uses index servers to partition the document index. Our previous work (Aboutabl 2013) presents a model that exploits parallelism on both shared-memory and cluster-based parallel architectures using term-based partitioning. We show that our shared-memory model outperforms the cluster-based model using some performance metrics.



(a) 3 x 3 2D Mesh (b) Linear Array

Figure 1. Example 2D Mesh and Linear Array Architectures

3. Parallel Query Processing on a 2D Mesh

In the proposed algorithm (figure 2), queries are fed into the 2D mesh from the left and propagated in parallel to the right until a query resides in every PE. All PEs in a column of the mesh receive input from and send output to adjacent PEs synchronously. The SIMD control model is adopted where all PE's perform the same operation simultaneously but on different data. In this sense, every processing element $P_{i,j}$ sends its current query to $P_{i,j+1}$.

```
propagate_queries
 Begin
    for j=0 to N-1 do
        for i=0 to N-1 do in parallel
            Pi,j send its current query to Pi,j+1
 End

propagate_keywords
```



```
 begin
    repeat
        for all processors P_i,,j do in parallel
           begin
             Check current query against the current keyword in P_i,j
             Send the current keyword in P_i,,j to P_i+1,j
           end
        Interchange keyword groups in mesh columns
     until all columns have been interchanged
  end

Mesh_QP
begin
for Q/N^2 groups of queries do
  begin
    Propagate_Queries
    for K/N^2 groups of keywords do
      Propagate_Keywords
  end
end
```

Figure 2. 2D Mesh query processing

In the same manner, keywords are propagated across the mesh rows in a top-down fashion. Keywords are fed into the vertical inputs of the first row and are sent from every $P_{i,j}$ to $P_{i+1,j}$ until each PE holds a keyword. Upon receiving a keyword, each PE checks its current query against its current keyword. If one of the keywords in the query matches the keyword, document ID's related to the keyword are appended to the list of document IDs of this query. Then, the current keyword is sent to the PE in the south. It is to be noted here that a query resides in a specific PE while keywords keep moving across the queries in columns. By the time N^2 keywords fill the mesh, every N keywords in a column would have been checked against all queries in the same column only. Therefore, keywords are propagated in the N columns interchangeably such that all the N^2 queries are checked against N^2 keywords.

The total number of queries Q is divided into groups each consisting of N queries. For every N query groups, each query group is propagated one query at a time through the N horizontal inputs as shown in figure 3. Hence, the mesh will handle N^2 queries at a time which are divided into N groups each consisting of N queries. Similarly, keywords are divided into groups of N keywords each where each group is fed into a column. Therefore, the mesh will check N^2 queries against N^2 keywords.

4. Parallel Query Processing on Linear Arrays

Two algorithms are proposed to perform parallel query processing on linear arrays. The first algorithm named LA_QPAKP (Linear Arrays Query Processing with All Keywords Propagation) relies on partitioning queries into groups. Each group of queries in the linear array is checked against all keywords which are propagated through the linear array passing through all PEs. On the other hand, our second algorithm named LA_QPKE (Linear Arrays Query Processing with Keywords Exchange) partitions both queries and keywords into groups where a group of queries is checked against a group of keywords. Keywords in every group are exchanged in and odd-even manner to ensure that all queries are checked with all keywords. Both algorithms are detailed in the following subsections. Here, N refers to the total number of PEs in the linear array.

4.1 LA_QPAKP

Queries and keywords are propagated through the linear array, each is input from a different end (figure 4). Queries are divided into Q/N groups of N queries each where each group is propagated separately through the linear array. After each query in a group settles in a PE, keywords propagation is started (figure 5). Here, keywords are not divided into groups but propagated in parallel as one stream till the last keyword k_{K-1} reaches P_0. Every propagation step involves sending the current keyword in each processor P_i to P_{i-1}; the next one in the direction of propagation. After each propagation step, all PEs check their current queries against their current keywords, in parallel. When all

the keywords have been propagated to the other end of the linear array, all queries in the current group of queries will have been checked against all keywords. Then, the next group of queries is fed in and all keywords are propagated again. This process is repeated for all query groups.

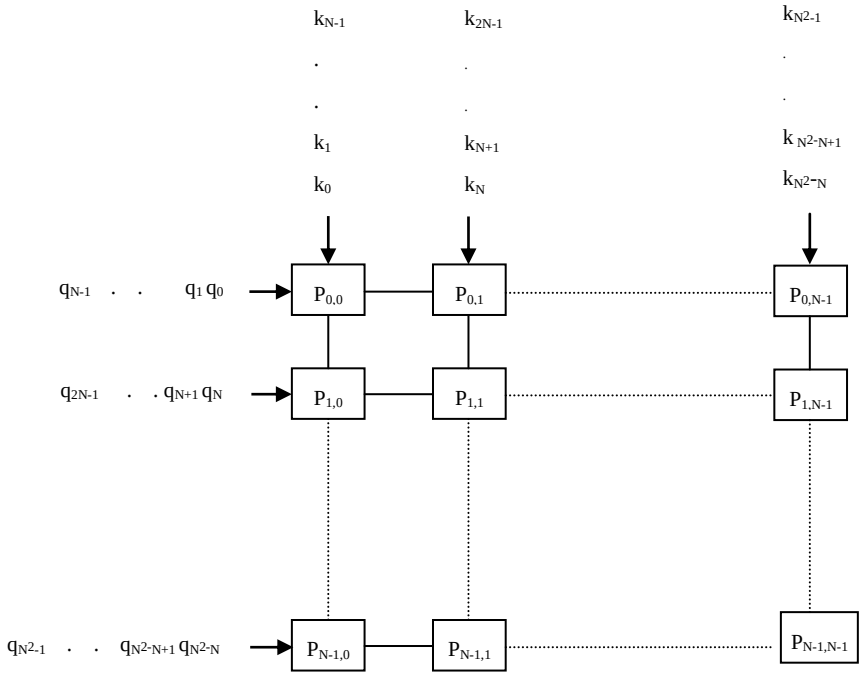


Figure 3. The distribution of N^2 queries and N^2 keywords among the inputs of an N ×N mesh

```
LA_QPAKP
begin
repeat
  for i=0 to N-1 do in parallel
     Pi sends its current query to Pi+1
  repeat
    for i=N-1 to 1 do in parallel
      begin
         Pi sends its current query to Pi-1
         Check current query against the current keyword in Pi
      end
  until the last input keyword reaches P0
until  all query groups are finished
end
```

Figure 4. Linear array query processing with all keywords propagation (LA_QPAKP)

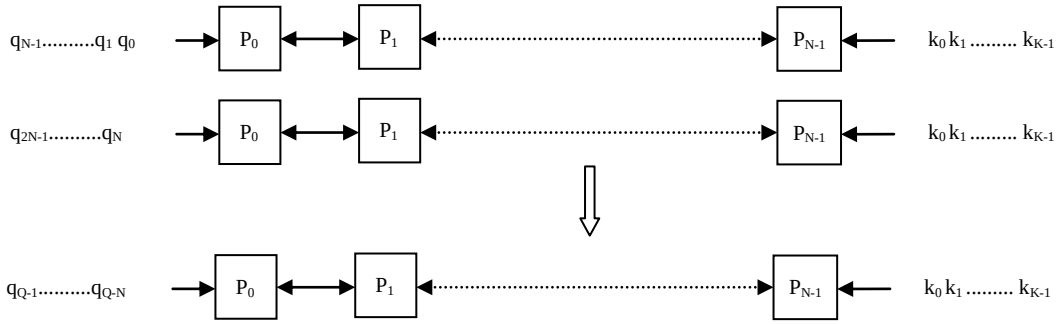


Figure 5. Sequence of propagation steps of queries and keywords in a linear array of N PEs using LA_QPAKP

4.2 LA_QPKE

Queries are divided into Q/N group of queries. Each group is propagated through the linear array till all N queries reside in the N PEs. Then, each group of the K/N groups of keywords is propagated and each keyword settles in one of the PEs. Hence, all keyword groups are propagated for every query group (figures 6 and 7). Once a group of N queries and a group of N keywords have settled in the linear array, all query-keyword pairs in all PEs are checked in parallel. To ensure that all keywords in the current keyword group are checked against all queries in the current query group, keywords are exchanged among the PEs in an odd-even manner. Every even-numbered PE exchanges its keyword with the next PE in a circular fashion; the first and last PEs may exchange keywords. This exchange is performed in parallel. Then, every query-keyword check takes place in parallel. The same steps are repeated with odd-numbered PEs. It takes N/2 steps of odd-even exchange with checking to finish comparing a query group to a keyword group.

```
LA_QPKE
begin
repeat
  for i=0 to N-1 do in parallel
     Pi sends its current query to Pi+1
  repeat
    for i=N-1 to 1 do in parallel
       Pi sends its current query to Pi-1
    for i=1 to N/2 do
       begin
         for all processors Pj do in parallel
            Check current query against the current keyword in Pj
         for all even numbered processors Pj do in parallel
            Pj exchanges keywords with P(j+1)mod N
         for all processors Pj do in parallel
            Check current query against the current keyword in Pj
         for all odd numbered processors Pj do in parallel
            Pj exchanges keywords with P(j+1)mod N
       end
   until all K/N keyword groups are finished
until all Q/N query groups are finished
end
```

Figure 6. Linear array query processing with keywords exchange (LA_QPKE)

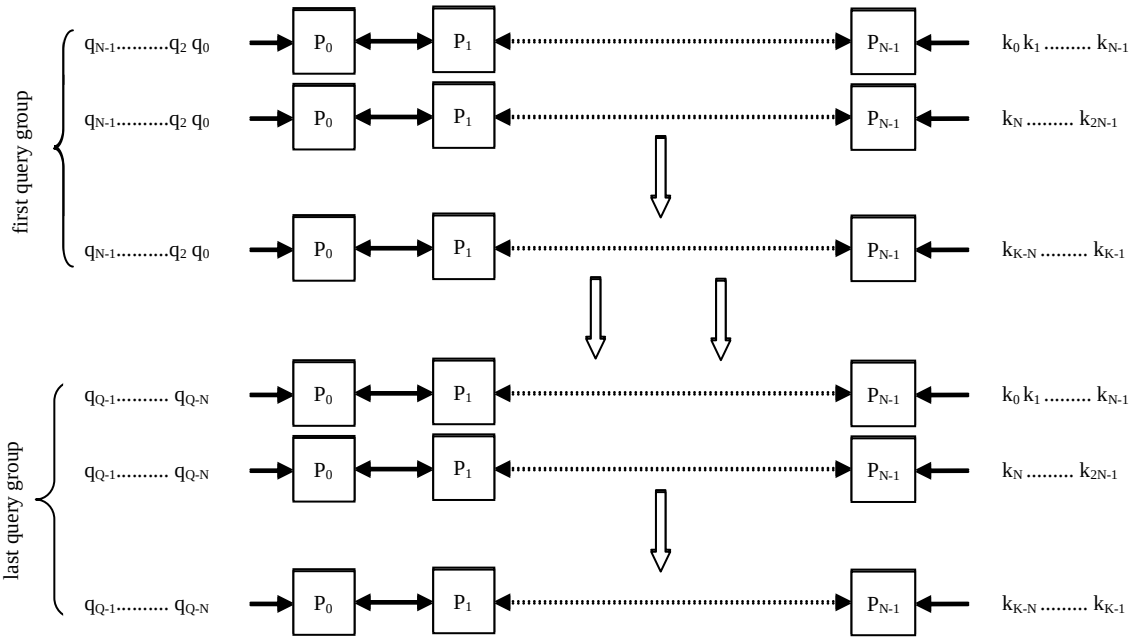


Figure 7. Sequence of propagation steps of groups of queries and keywords in a linear array of N PEs using LA_QPKE

5. Cost Model

The time cost of each of the proposed models is expressed in terms of the number of communication and computation steps. The cost of inter-processor communication greatly affects the efficiency of parallel algorithms. In this context, a communication step represents a transfer (send and receive) operation for a query or keyword from one processor to one of its neighbors. A computation step represents a comparison of a query against a keyword. In general, the total time cost, *T*, is computed as:

$$T = T_{comm} + T_{proc} \quad (1)$$

where T_{comm} is the total time spent in transferring queries and keywords along the links of the 2D mesh or linear array and T_{proc} is the total time spent in checking queries against keywords. Furthermore, T_{comm} and T_{proc} are computed as:

$$T_{comm} = t_{comm} \times s_{comm} \quad (2)$$

$$\mathrm{T_{proc}} = \mathrm{t_{proc}} \times \mathrm{s_{proc}} \quad (3)$$

where t_{comm} is the time needed for each communication step, s_{comm} is the number of communication steps, t_{proc} is the time needed for each comparison (processing) step and s_{proc} is the number of processing (comparison) steps. This general cost model is projected onto the query processing models proposed previously in this paper.

5.1 Two-dimensional Mesh Algorithm

In the proposed 2D mesh model for query processing, all PEs perform the comparison of a query against a keyword in parallel. Therefore, assuming a square $\mathrm{N \times N}$ mesh, every $\mathrm{N^2}$ comparisons consume one processing (comparison) step. As the total number of needed comparisons is KQ, the total number of computation steps, s_{proc}, is computed as :

$$s_{proc} = \frac{KQ}{N^2}$$

(4)

In terms of communication, every N^2 queries require:

1. N communication steps to propagate through the mesh and settle in their final destination PEs.
2. K communication steps for the K keywords to propagate through the queries' PE. The N-keyword groups keep propagating through the columns interchangeably in the N columns thus consuming N^2 communication steps i.e. N^2 keywords consume N^2 communication steps hence K keywords consume K communication steps.

Hence, the number of communication steps for N^2queries is $N + K$ and the total number of communication steps s_{comm} is

$$s_{comm} = \frac{Q(N+K)}{N^2}$$

(5)

Therefore, the total time can be modeled using equations (1) to (5) as :

$$T = \frac{t_{proc}KQ + t_{comm}Q(N+K)}{N^2}$$

(6)

5.2 Linear Array Algorithms

5.2.1 LA_QPAKP

As N queries propagate through the linear array in one direction, they require N communication steps. The K keywords require $K + N - 1$ communication steps to propagate from one end of the linear array till the last keyword reaches the other end. Therefore, for all queries Q divided into Q/N groups:

$$s_{comm} = \frac{Q}{N}(2N + K - 1)$$

(7)

$$s_{proc} = \frac{Q}{N}(K + N - 1)$$

(8)

From equations (1-3),(7) and (8), the time for LA_QPAKP can be modeled as :

$$T = \frac{Q}{N}\left(t_{proc}(K + N - 1) + t_{comm}(2N + K - 1)\right)$$

(9)

5.2.2 LA_QPKE

As queries are divided into Q/N groups, a group of N queries propagates through the linear array consuming N communication steps. For each query group, keywords are similarly divided into K/N groups, each one propagated through the linear array in N communication steps. Odd and even exchange of keywords among neighboring processors is also performed in N communication steps. For each keyword group, N computation steps are needed. Hence, the following equations are obtained:

$$s_{comm} = \frac{Q}{N}(N + 2K) \tag{10}$$

$$s_{proc} = \frac{KQ}{N} \tag{11}$$

$$T = t_{proc}\frac{KQ}{N} + t_{comm}\frac{Q}{N}(N + 2K) \tag{12}$$

5. Performance Evaluation

The proposed algorithms are evaluated using the speedup and efficiency performance measures. Speedup S_N measures the extent improvement in execution time and is computed as:

$$S_N = \frac{T_1}{T_N} \tag{13}$$

where T_1 and $T_N$are the sequential and parallel execution time respectively. Sequential execution requires KQ computation steps. Efficiency E_N is a measure of how much the processors are utilized.

$$E_N = \frac{S_N}{N} \tag{14}$$

Figures 8 and 9 show the speedup and efficiency results obtained from the three proposed algorithms. The 2D Mesh_QP algorithm outperforms the two linear array algorithms in terms of speedup and efficiency especially as the number of processors is increased. The 2D arrangement of processors in a 2D mesh as well as the 4-neighbour connectivity among processors allows for better exploitation of parallelism and hence higher scalability. On the other hand, both linear array algorithms, LA_QPKE and LA_QPAKP achieve less speedup and efficiency. LA_QPKE scales better than LA_QPAKP. The process of propagating all keywords across all queries in the linear array PEs using LA_QPAKP causes only slight improvement in speedup as the number of processors increases. Efficiency of 2D Mesh_QP is almost stable due to the scalability of the algorithm and is also higher than both LA_QPKE and LA_QPAKP. For both linear array algorithms, efficiency degrades as the number of processors increases. The rate of degradation is higher with LA_QPAKP as the keywords have to be propagated along a longer path for larger number of processors. The outperformance of LA_QPKE over LA_QPAKP, even though both are to be executed on the same type of architecture, is due to the way in which keywords are circulated among queries. LA_QPKE achieves this by local communication between neighbor PEs using odd and even exchange of keywords on a group of keywords. This communication pattern as well as keywords partitioning achieve better results when linear arrays are used.

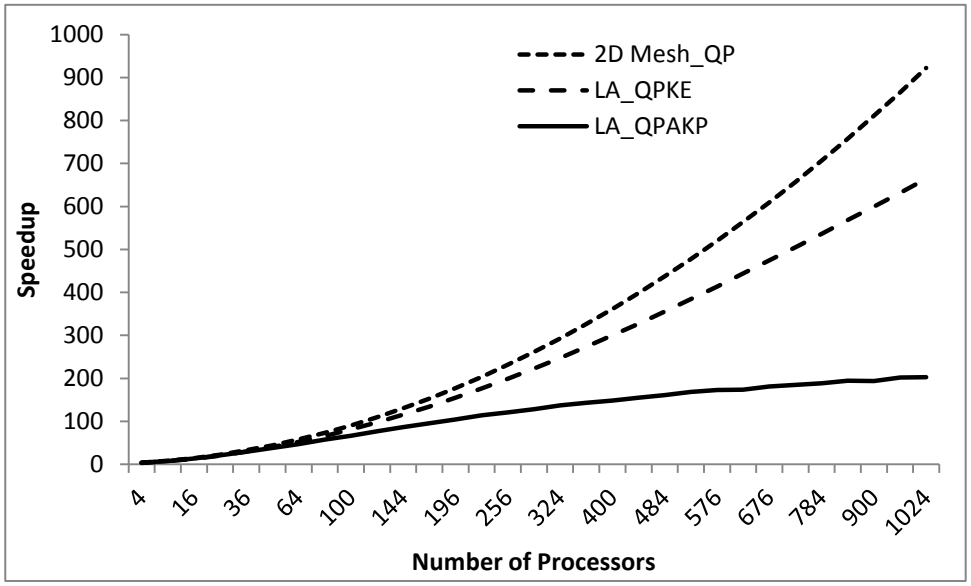


Figure 8. Speedup on varying the number of processing elements for 2D mesh and linear array algorithms for the same number of queries and keywords.

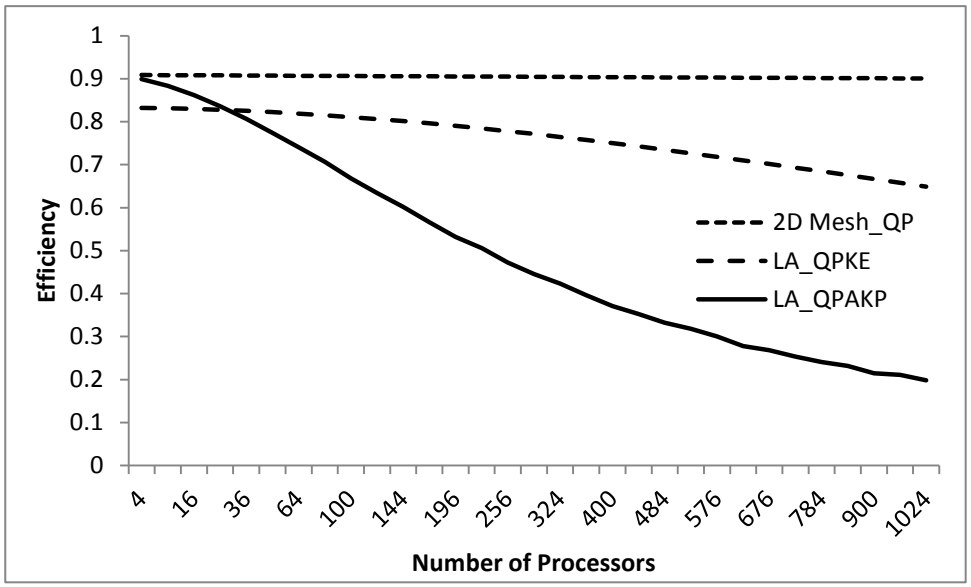


Figure 9. Efficiency on varying the number of processing elements for 2D mesh and linear array algorithms for the same number of queries and keywords.

7. Conclusion

The use of parallelism in search engines for the purpose of query-keywords comparison has become indispensable particularly with high query traffic and huge amount of documents on the web nowadays. This paper is an attempt to utilize the 2D mesh and linear array architectures to perform parallel query-keywords comparison. Three algorithms

are presented; one for 2D mesh architecture (2D Mesh_QP) and two for linear array architectures (LA_QPAKP and LA_QPKE). We attempt to exploit the arrangement of processors and the communication pattern in both types of architectures to achieve high speedup and efficiency. A cost model is presented for the three algorithms based on both processing and communication cost; equations (1) thru (14). Results show that in terms of speedup and efficiency performance metrics, 2D Mesh_QP outperforms both linear array algorithms for the same number of processors.

References

Aboutabl, A. E. 2013, "Exploiting Parallelism in Query Processing for Web Document Search Using Shared-Memory and Cluster-Based Architectures", *Computer and Information Science* **6**(3), 125-137.

Baeza-Yales, R., & Ribeiro-Neto, B. (1999), *Modern Information Retreival*, 1st Ed., Addison Wesley, Longman.

Büttcher, S., Clarke, C. L. A. & Cormack, G. V. (2010), *Parallel Information Retrieval* , In Information Retreival:Implementing and Evaluating Search Engines, MIT Press, 492-510.

Blelloch, G. E. & Maggs, B. M. (1996), "Parallel Algorithms", *Communications of the ACM.* **39**, 85-97.

Ceterchi, R., Pérez-Jiménez, M. J. & Tomescu, A. I. (2007), "Simulating the Bitonic Sort Using P Systems", WMC 2007 *Lecture Notes in Computer Science* **4860**, Springer, 172-192. doi: 10.1007/978-3-540-77312-2_11.

Cho, J. & Garcia-Molina, H. (2002), "Parallel crawlers", *Proceedings of the 11th international conference on World Wide Web*, 124-135. doi:10.1145/511446.511464.

Choi,Y. & Park, P. (2012), "Multicore and Mesh Network-based Parallel Performance Evaluation using Intra Prediction Algorithms", *International Journal of Control and Automation* **5** (4).

El-Rewini, H. & Abd-El-Barr, M. (2005), *Advanced Parallel Architectures and Parallel Processing*.Wiley Interscience.

Ercal, F. & Lee, H. C. (1997), "Time-Efficient Maze Routing Algorithms on Reconfigurable Mesh Architecture", *Journal of Parallel and distributed Computing* **44**(2), Elsevier, 133-140.

Grama, A., Gupta, A. & Kumar, G. V. (2003), *Introduction to Parallel Computing*. 2nd Ed. Pearson.

Hong, W. & Stonebraker, M. (1991), "Optimization of Parallel Query Execution Plans in XPRS". In *Proceedings of the First International Conference on Parallel and Distributed Information Systems, PDIS 1991*, 218-225. http://dx.doi.org/10.1109/PDIS.1991.183106.

Konstantopoulos, C., Mamalis, B., Pantziou, G. & GavalasHong, D. (2009), "Efficient parallel Text Retrieval techniques on Bulk Synchronous Parallel (BSP)/Coarse Grained Multicomputers (CGM)", *Journal of Supercomputing* **48**(3), Springer, 286-318. http://dx.doi.org/10.1007/s11227-008-0225-x.

Marin, M., Gil-Costa, V., Bonacic, C., Baeza-Yates, R. & Scherson, I. D (2010), "Sync/Async parallel search for the efficient design and construction of web search engines", *Parallel Computing* **36** (4), Elsevier, 153-168. doi:10.1016/j.parco.2010.02.001

Perhami, B. (2002), *Introduction to Parallel Processing: Algorithms and Architectures*, Kluwer Academic Publisher.

Panitanaraka, T. & Shontza, S. M. (2011), "MDEC: MeTiS-based Domain Decomposition for Parallel 2D Mesh Generation", *Procedia Computer Science* **4**, 302–311. doi:10.1016/j.procs.2011.04.032.

Young-Joo, S. & Yalamanchili, S. (2000), "Configurable algorithms for complete exchange in 2D meshes", *IEEE Transactions on Parallel and Distributed Systems* **11**(4), 337 – 356. doi: 10.1109/71.850832.