Parallelizing Breadth First Search Using CELL Broadband Engine

Rahul kumar Gayatri,Pallav Kumar Baruah
Sri Satya Sai University
Prashanthi Nilayam - 515134, Andhra Pradesh, India
{rahulgayatri84, baruahpk }@gmail.com

Abstract

Multicore processors are a shift of paradigm in computer architecture that promises dramatic increase in performance. But they also bring complexity in algorithmic design. In this paper we describe the challenges and design issues involved in parallelizing Breadth First Search on Cell Brodband Engine (Cell BE).We divide the graphs into subgraphs and assign them to different processing elements.The major advantage of this algorithm is that we can optimize on interprocessor communication.

1 Introduction

The Cell BE processor is a joint venture of IBM,Sony and Toshiba. It is a heterogeneous multicore processor and has been of much importance in High Performance Computing due to the high flop rates it provides. It consists a PowerPc core (PPE) which controls eight SIMD cores called Synergistic Processing Elements (SPE's) which are computational powerhorses. Programming on Cell processor is a challenge due to the limited memory available in the SPE's (256 KB). Hence applications need significant changes to fully exploit the novel architecture. Many areas of science (astrophysics,artificial intelligence,national security) need techniques to explore large scale data sets which are in the form of graphs. Among graph search algorithms Breadth First Search (BFS) is most important for graph analysis applications. The problem of searching large graphs is due to the vast search spaces, especially in Cell BE due to limited memory available in its processing elements. In most applications search algorithms are used to discover vertices's and paths. The sequential algorithm is explained by Oreste et.al.[1] where they have given an algorithm for parallelizing BFS by distributing vertices's among different processors.This algorithm addresses the issue of load balancing, but it has a drawback of issuing more number of DMA's.In our algorithm we propose to optimize on the DMA calls.

2 Breadth First Search Algorithm (BFS)

In this section we present the methodology to parallelize the breadth first search algorithm. We will first introduce the notation we will follow and a sequential version of BFS. A graph G = (V,E) is a set of vertices's V and edges E. The size of the graph is given by $|V|$. Given a vertex $v \in V$, $E_v = \{\mathrm{w} \in V : (v, w) \in E\}$ and vertex arity i. e. , the number of nodes adjacent to the vertex v is given as $|E_v|$. Given a graph G = (V,E) and the root vertex $r \in V$

the BFS algorithm explores the edges of G to find all the vertices's reachable from r and produces an array named LEVEL of the size $|V|$, where $LEVEL_v$ gives the level of vertex v i. e. , number of edges that need to be traversed from r to reach v.

In any level, Q is the set of vertices's that must be visited at the current level and Q_{next}, the set of vertices's to be visited in the next level i. e. , vertices's adjacent to those in Q. At the end of exploration of a level, Q_{next} is copied to Q and Q_{next} emptied. The algorithm terminates when at the end of exploration of some level there are no vertices's left in Q_{next} i. e. , there are no more vertices's to be explored. The algorithm visits a vertex only once. To do so it maintains an array of marked variable $marked_v$ indicating tells whether vertex v has already been visited. Adjacent vertices's are added to Q_{next} only if that vertex is not marked. Also there is a level array which maintains the information about the level of the vertices's visited.

```
Algorithm 1 Sequential BFS exploration of a graph.
Input:      $G(V,E)$, graph;
            $r$,      root vertex;
Variables:  $level$,  exploration level;
            $Q$,      vertices to be explored in the current level;
            $Qnext$,  vertices to be explored in the next level;
            $marked$, array of booleans: $marked_i \ \forall i \in [1...|V|]$;

$\forall i \in [1...|V|] : marked_i$ = false
$marked_r$ = true
$level \leftarrow 0$
$Q \leftarrow \{r\}$
repeat
    $Qnext \leftarrow \{\}$
    for all $v \in Q$ do
        for all $n \in E_v$ do
            if $marked_n$ = false then
                $marked_n \leftarrow$ true
                $Qnext \leftarrow Qnext \cup \{n\}$
            end if
        end for
    end for
    $Q \leftarrow Qnext$
    $level \leftarrow level + 1$
until $Q = \{\}$
```

Figure 1: **Sequential algorithm for Breadth First Search**

The most straightforward way to parallelize the algorithm is to explore the vertices's concurrently with all the processing elements. In Cell BE a part of Q will be given to each of the SPE's. The SPE's will then explore these vertices's, mark them and send their adjacent vertices's to PPE. These nodes will become Q_{next} in the PPE. At the end of each iteration, all the nodes of Q_{next} will be copied to Q and the process repeated. The drawback of this method is that at the end of each iteration PPE has to again distribute the work among different SPE's apart from copying Q_{next} to Q. Also on the marked array an exclusive lock has to be placed so that more than one processing element does not access the array at any given time. One of the main drawbacks of parallel computing is the communication overhead involved between different processing elements.We will minimize on these factors.

In our algorithm we adopt a different approach from [1]. We store the adjacent vertices's of a given node in the form of an array. When a DMA is issued for the adjacent vertices's of a given vertex, all of them can be brought into Q_{next} in a single DMA. This will drastically reduce the number of DMA's issued in getting the adjacent vertices's into Q_{next}. Associated with each SPE i we have a Q_i and Q_{next_i} in PPE. Once the vertices's are distributed among the different SPE's, the subgraphs with roots as these vertices's are all marked by the same SPE. In this way the only time an spe needs to communicate with PPE is at the end of each iteration when Q_{next_i} of a SPE has to be copied to its corresponding Q_i. The nodes of Q_{next_i} and Q_i have to be stored in PPE because of the limited space available in SPE. For large graphs this may not be sufficient to store all nodes for that respective SPE.

3 Implementation of the algorithm

Each vertex is a structure and contains
1) a pointer to an array of its adjacent ver-

tices's
2) an integer which gives the attributes of the vertex.
3) vertex identifier.
Each vertex is of size 16 bytes so that it is compatible with DMA size and a maximum of 1024 vertices's can be transferred in a single DMA. Input is of the form of an array of size $|V|$ and type vertex.
The different phases of this implementation are as follows

START : PPE initiates the computation by placing the root vertex in Q marking it with level 0 and getting vertices's adjacent to root vertex into Q_{next} It then distributes the work among available SPE's and informs the SPE.
FETCH : Once the SPE receives message from PPE it makes a DMA for the nodes that are assigned to it and marks them.
GATHER : It then explores the nodes and gets their adjacent nodes into Q_{next} if they are not already marked.
REPEAT : After all the vertices's in the present level are explored it copies Q_{next} into Q and repeats marking and GATHER step until it has no more vertices's to explore.
OUTPUT : After exploring all the vertices's assigned to it, it writes back all the vertices's marked by it to PPE.

But for large graphs the SPE memory may not be sufficient to hold all the vertices's in Q and Q_{next} Hence the vertices's have to be stored in PPE and brought into SPE in blocks. Let us assume bQ is the portion of Q that is fetched in each DMA transfer and bQ_{next} is the portion of Q_{next}. Corresponding to each SPE there is a Q_{next_i} and Q_i in PPE.

Revised Algorithm with storage constraints

START : PPE initiates the computation, marks the root vertex and puts all vertices's adjacent to root in to Q_{next}. Instead of distributing the work it continues the work by copying Q_{next} into Q and exploring them until it has sufficient vertices's so that their exploring overshadows the DMA latency incurred in distributing vertices's. It then copies the vertices's corresponding to each SPE into its respective Q_i and sends a message to the SPE's.

FETCH : Once the SPE receives message from PPE it fetches bQ_i a portion of Q_i into its local buffer in a double buffering fashion. This means there are 2 data structures associated with bQ_i. We need to wait only for the first DMA transfer to complete and we can swap buffers and start a new transfer for the next block of data from Q_i. We can make the data arrived in the first buffer available for marking and subsequent steps. Marking vertices is done by changing the last 8 bits of integer containing vertex attributes.

GATHER : This step explores the vertices's in bQ_i and loads their respective adjacent arrays into bQ_{next_i} until it is full by using DMA calls. For this it is necessary to know the size of the adjacency array and here the information stored in each vertex about the length of its adjacency array is used. Adjacency arrays are put into bQ_{next_i} only if they are not already marked. It is enough if we check the first element of the adjacency array since if it is marked then all are marked. The integer containing the attributes of the vertex contains 2 fields 1) length of the adjacency array of the vertex 2) level of the vertex. Once bQ_{next_i} is full they are copied back into PPE into their respective Q_{next_i}. This step is also done in a double buffering fashion i. e. ,there are two structures associated with bQ_{next_i}.

REPEAT : Once all the nodes in Q_i are explored SPE informs PPE that it has finished exploring all nodes in that particular level. PPE then checks into Q_{next_i} of each SPE if there are vertices's left to be explored by their respective SPE's if yes it copies Q_{next_i} to Q_i and informs SPE to start computation again. Else if there are no more vertices's left to be explored by that SPE it informs it to terminate the program.

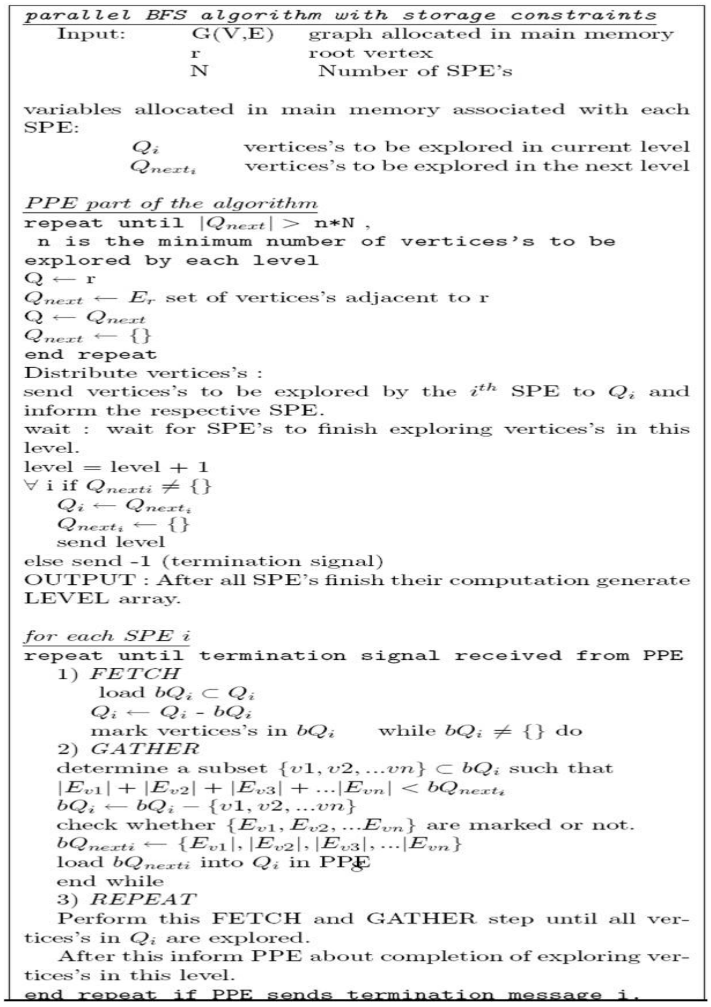
OUTPUT : Once all the SPE's have finished their computation PPE scans through all the vertices's and generates the LEVEL array by using vertex identifier as the index and level of that vertex as its value in the array.

4 Performance

Performance in parallel algorithms is measured in terms of accuracy of the results and speed of execution of the algorithm. In our case it is to calculate least distance of each vertex from root and also the time it takes to mark all vertices's.The method we presented is optimal for graphs whose average arity of vertices's are almost same. Therefore marking the subgraphs given to each of the SPE's will take almost same number of iterations. This method will give better results than those in [1]. The method presented in [1] involves a lot of communication overhead because of the messages it passes between SPEs at the end of each iteration. Hence we have a more optimized algorithm as the SPE's need not interact with each other as they mark separate subgraphs.

5 Conclusion

Along with increase in performance, multi-core processors add complexity in software development. The complexity is due to many activities running concurrently. A significant factor effecting parallel algorithms performance is the inter-processor communication. In our case it is the DMA's between SPE and PPE. In our algorithm DMA latency can be overshadowed by performing work of marking level in the FETCH stage and checking whether vertices's are marked or not in the GATHER stage.



```
parallel BFS algorithm with storage constraints
    Input:      G(V,E)     graph allocated in main memory
                r          root vertex
                N          Number of SPE's

variables allocated in main memory associated with each
SPE:
            $Q_i$          vertices's to be explored in current level
            $Q_{next_i}$   vertices's to be explored in the next level

PPE part of the algorithm
repeat until $|Q_{next}| >$ n*N ,
 n is the minimum number of vertices's to be
explored by each level
$Q \leftarrow r$
$Q_{next} \leftarrow E_r$ set of vertices's adjacent to r
$Q \leftarrow Q_{next}$
$Q_{next} \leftarrow \{\}$
end repeat
Distribute vertices's :
send vertices's to be explored by the $i^{th}$ SPE to $Q_i$ and
inform the respective SPE.
wait : wait for SPE's to finish exploring vertices's in this
level.
level = level + 1
$\forall$ i if $Q_{nexti} \neq \{\}$
    $Q_i \leftarrow Q_{next_i}$
    $Q_{next_i} \leftarrow \{\}$
    send level
else send -1 (termination signal)
OUTPUT : After all SPE's finish their computation generate
LEVEL array.

for each SPE i
repeat until termination signal received from PPE
    1) FETCH
         load $bQ_i \subset Q_i$
        $Q_i \leftarrow Q_i - bQ_i$
        mark vertices's in $bQ_i$      while $bQ_i \neq \{\}$ do
    2) GATHER
    determine a subset $\{v1, v2, ...vn\} \subset bQ_i$ such that
    $|E_{v1}| + |E_{v2}| + |E_{v3}| + ...|E_{vn}| < bQ_{next_i}$
    $bQ_i \leftarrow bQ_i - \{v1, v2, ...vn\}$
    check whether $\{E_{v1}, E_{v2}, ...E_{vn}\}$ are marked or not.
    $bQ_{nexti} \leftarrow \{|E_{v1}|, |E_{v2}|, |E_{v3}|, ...|E_{vn}|\}$
    load $bQ_{nexti}$ into $Q_i$ in PPE
    end while
    3) REPEAT
    Perform this FETCH and GATHER step until all ver-
tices's in $Q_i$ are explored.
    After this inform PPE about completion of exploring ver-
tices's in this level.
end repeat if PPE sends termination message i.
```

Figure 2: **parallel BFS algorithm with storage constraints**

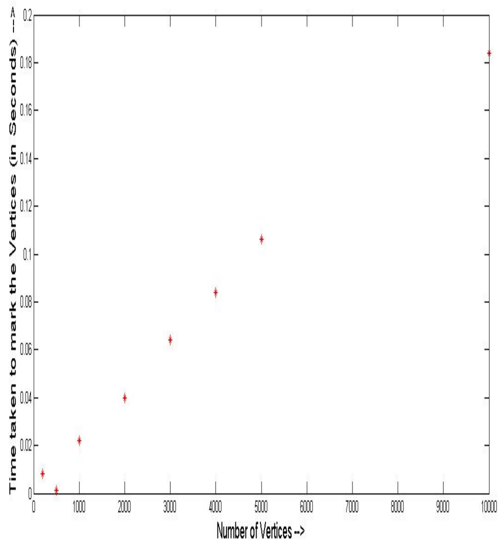


Figure 3: **Execution time of the Algorithm**

6 Future Work

We can select an application for which this algorithm can be used to find optimal solution and enhance the performance by reducing the running time of the application.Also we can find paths from the root to the solution.We can also add weight corresponding to each edge.

References

[1] Oreste Villa,Daniele Paolo Scarpazza,Fabrizio Petrini, Juan Fernandez Peinador.Challenges in Mapping Graph Exploration Algorithms on Advanced Multi-core Processors. *Parallel and distributed symposium, 2007 IPDPS.IEEE international*

[2] D.A Bader and k.Madduri.Designing Multithreaded Algorithms for Breadth First Search and st-connectivity on CRAY MTA-2.In *Proc. Intl Conf on Parallel Processing August 2006*

[3] J.Feo Optimized BFS algorithm on MTA-2 Architecture.Personal communication 2006

[4] Programming the Cell Broadband Engine Architecture Examples and Best Practices: *www.redbooks.ibm.com/redbooks/pdfs/sg247575.pdf*