THOMSON REUTERS
ENDNOTE

[Mustafa, 4(1): January 2017] ISSN 2348 – 8034
DOI- 10.5281/zenodo.263984 Impact Factor- 4.022

GLOBAL JOURNAL OF ENGINEERING SCIENCE AND RESEARCHES

THE BEST FIT IN MEMORY MANAGEMENT

Mohammed N. Mustafa*

ABSTRACT

Dynamic memory allocation often makes up a large part of program execution time. Different variants of the best-fit allocator are implemented and their space and time costs measured and compared. We found variants of this algorithm that are 3-33% faster than the Doug Lea 2.7.0 allocator

Keywords: *Transient, Fin, Conduction, Natural Convection, Finite element technique*

I. INTRODUCTION

Best Fit is the most powerful method in the Wisconsin Package(TM) for identifying the best region of similarity between two sequences whose relationship is unknown.

Best Fit makes an optimal alignment of the best segment of similarity between two sequences. Optimal alignments are found by inserting gaps to maximize the number of matches using the local homology algorithm of Smith and Waterman.

Best Fit inserts gaps to obtain the optimal alignment of the best region of similarity between two sequences, and then displays the alignment in a format similar to the output from Gap. The sequences can be of very different lengths and have only a small segment of similarity between them. You could take a short RNA sequence, for example, and run it against a whole mitochondrial genome.

Best Fit accepts two individual nucleotide sequences or protein sequences as input. The function of Best Fit depends on whether your input sequence(s) are protein or nucleotide. Programs determine the type of a sequence by the presence of either Type: N or Type: P on the last line of the text heading just above the sequence. If your input sequences are peptide sequences, this program uses a scoring matrix, bIosum62.cmp, with comparison values derived from a study of substitutions between amino acid pairs in ungapped block of aligned protein segments as measured by Henikoff and Henikoff (Proc. Natl. Acad. Sci. USA 89; 10915-10919 (1992)).

The Best Fit technique searches the list for the hole that best fits the process. It's slower than First Fit since the entire list must be searched. It also produces tiny holes that are often useless. "Best fit" tries to find a block that is "just right". The problem is that this requires keeping your free list in sorted order, so you can see if there is a really good fit, but you still have to skip over all the free blocks that are too small. As memory fragments, you get more and more small blocks that interfere with allocation performance, and deallocation performance requires that you do the insertion properly in the list.

Soagain, this allocator tends to have truly lousy performance In many ways, the most natural approach is to allocate the free block that is closest in size to the request. This technique is called best fit. In best fit, we search the list for the block that is smallest but greater than or equal to the request size. Like first fit, best fit tends to create significant external fragmentation, but keeps large blocks available for potential large allocation requests

II. ALGORITHM OF BEST FIT

- Goal
- Find the smallest memory block into which the job will fit
- Entire table searched before allocation.

[Mustafa, 4(1): January 2017]
DOI- 10.5281/zenodo.263984

ISSN 2348 – 8034
Impact Factor- 4.022

Table 1

These two snapshots of memory show the status of each memory block before and after a request is made using the best-fit algorithm.

Before Request		After Request	
Beginning Address	Memory Rock Size	Beginning Address	Memory Block Size
4075	105	4075	105
5225	5	5225	5
6785	600	6785	6oo
7560	20	7560	20
7600	205	7800	5
10250	4050	10250	4050
1525	230	1525	230
24500	1000	24500	1000

Best Fit uses the local homology algorithm of Smith and Waterman (Advances in Applied Mathematics 2; 482-489 (1981)) to find the best segment of similarity between two sequences. Best Fit reads a scoring matrix that contains values for every possible GCG symbol match. The program uses these values to construct a path matrix that represents the entire surface of comparison with a score at every position for the best possible alignment to that point. The quality' score for the best alignment to any point is equal to the sum of the scoring matrix values of the matches in that alignment, less the gap creation penalty times the number of gaps in that alignment, less the gap extension penalty times the total length of all gaps in that alignment. The gap creation and gap extension penalties are set by you. If the best path to any point has a negative value, a zero is put in that position.

After the path matrix is complete, the highest value on the surface of comparison represents the end of the best region of similarity between the sequences. The best path from this highest value backwards to the point where the values revert to zero is the alignment shown by BestFit. This alignment is the best segment of similarity between the two sequences.

For nucleic acids, the default scoring matrix has a match value of 10 for each identical symbol comparison and -9 for each non-identical comparison (not considering nucleotide ambiguity symbols for this example). The quality score for a nucleic acid alignment can, therefore, be determined using the following equation:

Quality = 10 x TotalMatches + -9 x TotalMismatches - (GapCreationPenalty x GapNumber) - (GapExtensionPenalty x TotalLengthOfGaps)

The quality score for a protein alignment is calculated in a similar manner. However, while the default nucleic acid scoring matrix has a single value for all non-identical comparisons, the default protein scoring matrix has different values for the various non-identical amino acid comparisons. The quality score for a protein alignment can therefore be determined using the following equation (where Total(AA) is the total number of A-A (Ala-Ala) matches in the alignment, CmpVal(AA) is the value for an A-A comparison in the scoring matrix, Total(AB) is the total number of A-B (Ala-Asx) matches in the alignment, CmpVal(AB) is the value for an A-B comparison in the scoring matrix, ...) :

Quality = CmpVal(AA) x TotaI(AA) + CmpVal(AB) x Total(AB) + CmpVal(AC) x Total(AC) + CmpVal(ZZ) x Total(ZZ) - (GapCreationPenalty x GapNumber) - (GapExtensionPenalty x TotalLengthOfGaps)

[Mustafa, 4(1): January 2017] ISSN 2348 – 8034
DOI- 10.5281/zenodo.263984 Impact Factor- 4.022

Best Fit Always Finds Something.

BestFit always finds an alignment for any two sequences you compare -¬even if there is no significant similarity between them! You must evaluate the results critically to decide if the segment shown is not just a random region of relative similarity.

The Segments Shown Obscure Alternative Segments

BestFit only shows one segment of similarity, so if there are several, all but one are obscured. You can approach this problem with graphic matrix analysis (see the Compare and DotPlot programs). Alternatively, you can run BestFit on ranges outside the ranges of similarity found in earlier runs to bring other segments out of the shadow of the best segment.

The Best Fit is Only One Member of a Family

Like all fast gapping algorithms, the alignment displayed is a member of the family of best alignments. This family may have other members of equal quality, but will not have any member with a higher quality. The family is usually significantly different for different choices of gap creation and gap extension penalties. See the CONSIDERATIONS topic in the entry for the Gap program in the Program Manual to learn more about how to assign gap creation and gap extension penalties.

III. DEFAULT GAP PENALTIES ARE SPECIFIC TO EACH SCORING MATRIX

Best Fit chooses default gap creation and extension penalties that are appropriate for the scoring matrix it reads. If you select a different scoring matrix with the -MATRix command-line parameter, the program will adjust the default gap penalties accordingly. You can use -GAPweight and -LENgthweight to specify alternative gap penalties if we don't want to accept the default values.

IV. RAPID ALIGNMENT

When possible, Best Fit tries to find the optimal alignment very quickly.If this rapid alignment is not unambiguously optimal, Best Fit automatically realigns the sequences to calculate the optimal alignment.When this occurs, the monitor of alignment progress on your terminal screen (Aligning...) is displayed twice for a single alignment.

To run BESTFIT, select the gb_ro:ml5034 and the gb_ro:ml9993 sequences from the Main list window, move your cursor to Functions and select Best Fit from the Pairwise Comparison Menu:

ISSN 2348 – 8034
Impact Factor- 4.022

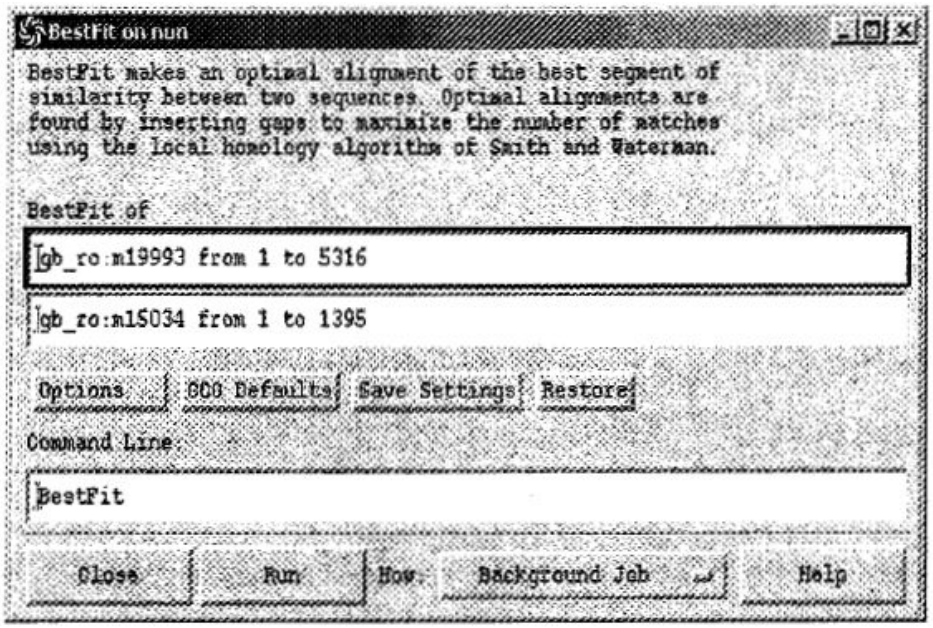

Figure: 1

.The Best Fit window will appear. Click on the Options button. The Option Window is very similar to the GAP Option window. From this window, select "Set thresholds...." and type "4" into the highlighted window, de-select"Abbreviate output.....", and select both "New sequence...". Change the name ofthe output file names by adding a "-1" behind the accession number (ex. M19993-l.gap). This will distinguish the GAP and Best Fit result sequences.

ISSN 2348 – 8034
Impact Factor- 4.022

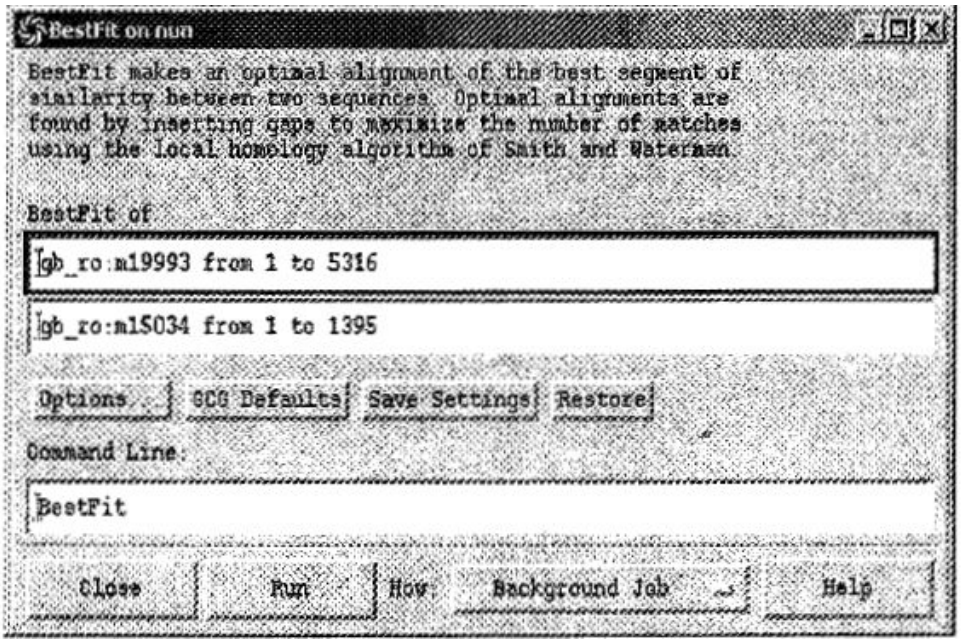

Figure: 2

[Mustafa, 4(1): January 2017]
DOI- 10.5281/zenodo.263984

ISSN 2348 – 8034
Impact Factor- 4.022

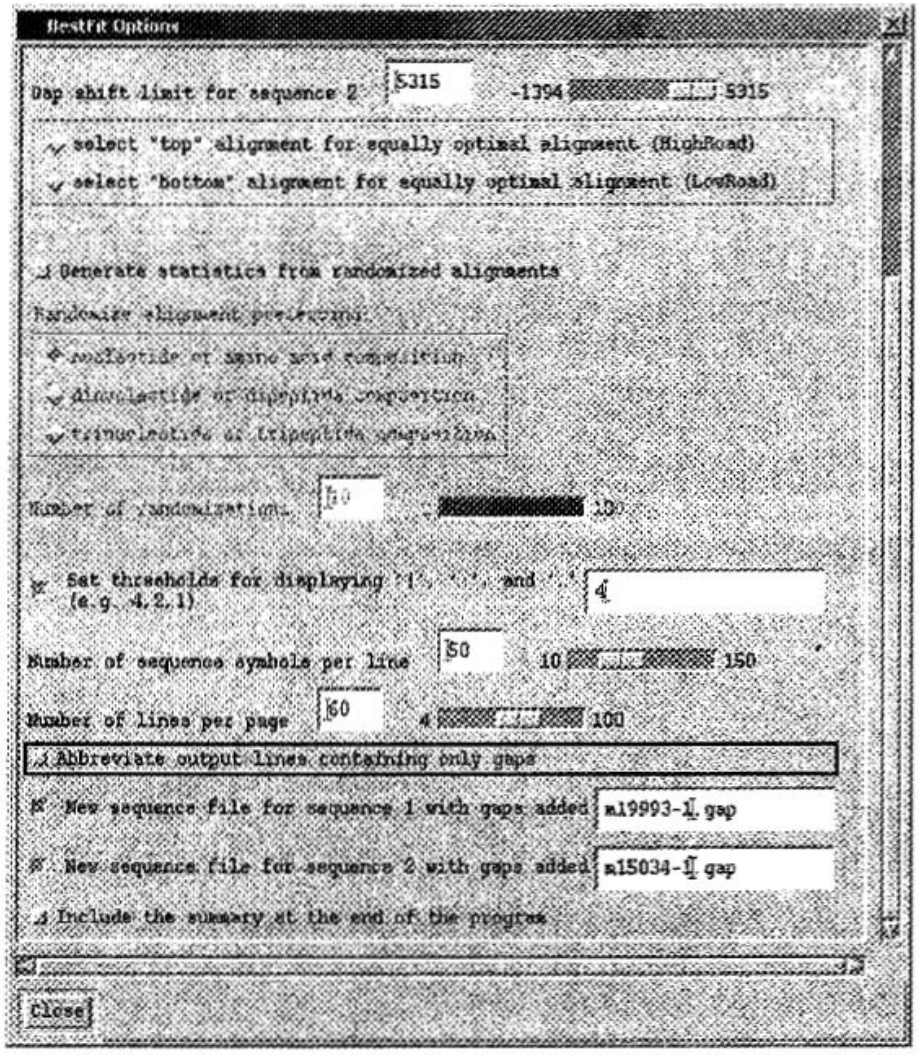

Figure: 3

The results obtained from the Best Fit program provides the user with the BEST alignment found. The best alignment may not contain the entire sequence; in fact it usually does not. One difference is that the ends of the genomic sequence have been left off the alignment. Only GAP will provide full length sequences in the output alignment.

Once again, for most alignments the default options will provide a reasonable alignment.

To view a sample output file, click here, (link to bestfit_output.txt)

The .gap files can be used by the GAPSHOW program to provide a graphical representation of the Best Fit results; Follow the instructions found in theGAPSHOW section to produce the following figure:

ISSN 2348 – 8034
Impact Factor- 4.022

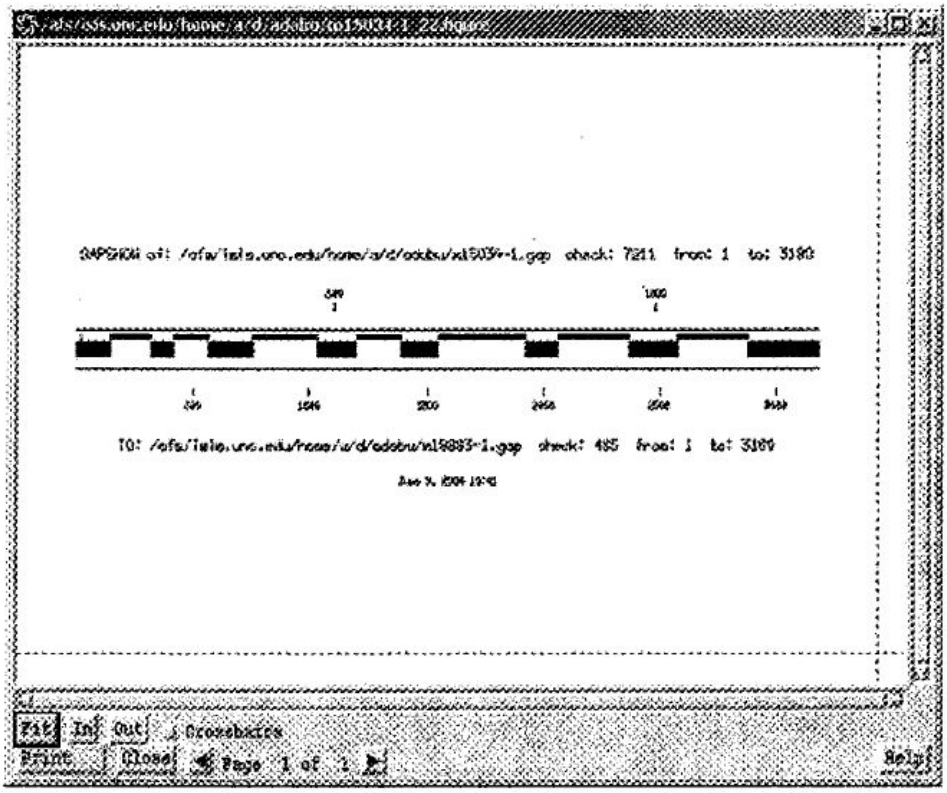

Figure: 4

V. BEST FIT ALLOCATION

Search for the most suitable partition for the job. The best choice is when partition size= job size. This may happen seldom, so the next choice is the smallest available partition but large enough to accommodate the job.

NOTE: If the memory' partitions are sorted in ascending order according to their size then the first fit = the best fit. bestfit works slow.it is complex.less memory wastage.time consuming.

Example: Best Fit Allocation
we show only the final four steps of best fit allocation for this example. The memory layouts for these requests is shown in table 2.

Table 2: Best Fit Allocation

A8	20	15	8	2	25	58		
A30	20	15	8	2	25	30	28	
D15	35		8	2	25	30	28	
A15	35		8	2	25	30	15	13

[Mustafa, 4(1): January 2017]
DOI- 10.5281/zenodo.263984

ISSN 2348 – 8034
Impact Factor- 4.022

VI. The Strategies That Allocate Space in Memory

A number of strategies are used to allocate space to the processes that are competing for memory.

- **Best Fit**

The allocator places a process in the smallest block of unallocated memory in which it will fit.

Problems:

- It requires an expensive search of the entire free list to find the best hole.
- More importantly, it leads to the creation of lots of little holes that are not big enough to satisfy any requests. This situation is called fragmentation, and is a problem for all memory-management strategies, although it is particularly bad for best-fit

Solution*:*
One way to avoid making little holes is to give the client a bigger block than it asked for. For example, we might round all requests up to the next larger multiple of 64 bytes. That doesn't make the fragmentation go away, it just hides it.

- Unusable space in the form of holes is called external fragmentation
- Unusable space in the form of holes is called external fragmentation

- **Worst Fit**

The memory manager places process in the largest block of unallocated memory available. The ides is that this placement will create the largest hole after the allocations, thus increasing the possibility that, compared to best fit, another process can use the hole created as a result of external fragmentation.

- **First Fit**

Another strategy is first fit, which simply scans the free list until a large enough hole is found. Despite the name, first-fit is generally better than best-fit because it leads to less fragmentation.

Problems:

- Small holes tend to accumulate near the beginning of the free list, making the memory allocator search farther and farther each time.

Solution:

- Next Fit

Next Fit
The first fit approach tends to fragment the blocks near the beginning of the list without considering blocks further down the fist. Next fit is a variant of the first-fit strategy.The problem of small holes accumulating is solved with next fit algorithm, which starts each search where the last one left off, wrapping around to the beginning when the end of the list is reached (a form of one-way elevator) is exactly the same.Best-Fit Versus First-Fit Allocation

Two methods for free space allocation:

***First-tit memory allocation*:**
first partition fitting the requirements

ISSN 2348 – 8034
Impact Factor- 4.022

.Leads to fast allocation of memory space

Best-fit memory allocation:
smallest partition fitting the requirements

. Results in least wasted space
. Internal fragmentation reduced, but not eliminated

Fixed and dynamic memory allocation schemes use both methods

Best-fit memory allocation
Advantage: makes the best use of memory space
Disadvantage: slower in making allocation

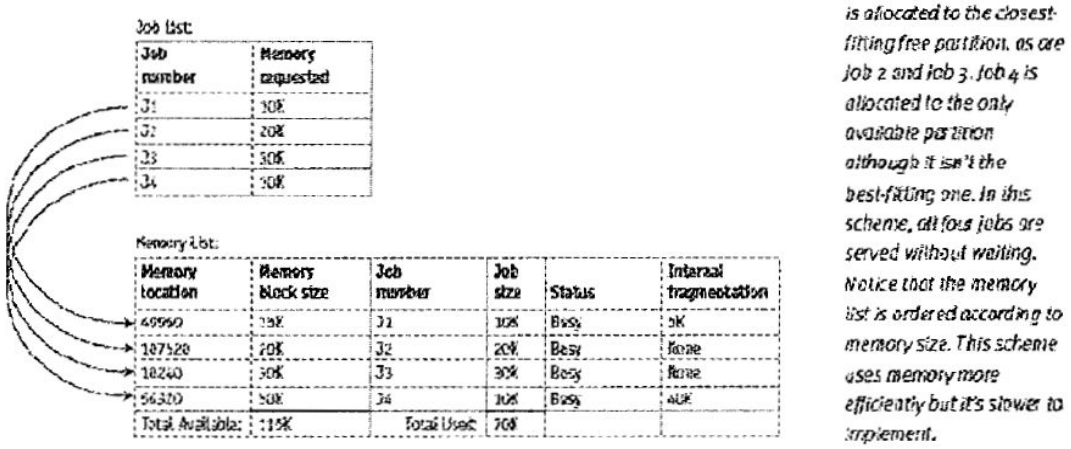

Figure: 5

VII. CODE OF BEST FIT ALGORITHM

```
Option Explicit

Dim mem(97 To 107) As Byte

Dim repair_mem(97 To 107) As Byte

Dim size, I As Integer

Private Sub Command1_Click()

Dim i, I As Byte

i = Asc(Textl.Text)

mem(i) = Val(Text2.Text)

Textl.Text = ""
```

ISSN 2348 – 8034
Impact Factor- 4.022

```
Text2.Text =""

End Sub

Private Sub Command2_Click()

Dim Job_Req, i, j, r, t, z, Space, Spacel, flag As Integer

Job_Req = Text3.

Text size = Text4.Text

For i = 97 To 97 + size

If (mem(i) = 0) Then

List2.Addltem Chr$(i) & "------>in use"

Else

List2.Addltem Chr$(i) & "------>" & mem(i) & "Hole"

End If

Next i

Space = 0

For i = 97 To 107

If (mem(i) >= Job_Req) Then

Space1 = mem(i) - Job_Req

If (Space1> Space) Then

Space = Space1

flag = i

End If

End If

Next i

For i = 97 To 97 + size

If (mem(i) = 0) Then

Listl.Addltem Chr$(i) & "------->in use"

Else

Listl.Addltem Chr$(i) & "------>" & mem(i) & "Hole"
```

ISSN 2348 – 8034
Impact Factor- 4.022

```
End If

Next i

Listl.Addltem "we use the address-->" & Chr$(flag) & "Best fit strategе"

End Sub

Private Sub Command3_Click()

End

End Sub
```

VIII. FIRST FIT AND BEST FIT MAPPINGS

In 1992, the FF and BF bit-maps (First Fit and Best Fit bit-maps) were introduced in order to solve the allocation miss problem in the FS strategy. In such bit-map approaches, a 2D-array system status (N = R x C) is used to store a free/busy status-bit for every processor. For an incoming request (r X c), all N bits have to be identified at least twice to find the corresponding available subsystem therefore, the time complexity of these bit-map methods is O(N). The bit-map strategies gave better system utilization than the FS strategy. However, these strategies did not have complete recognition capability since they did not provide task rotation.

IX. THE BUSY LIST STRATEGY

The best fit BL (Busy List) strategy was proposed in 1993 [4] and improved in 1996 [5]. This strategy improved time complexity as well as system fragmentation over the BF bit-map. It uses "a busy list" to store allocated sub-systems and proposes the "maximum boundary value (BV)" as the best-fit criteria. For an incoming request task (r X c), all (up to 8) candidate sub-systems of size S(r, c) or S(c, r) are created from each of the 4 corners of a particular allocated sub-system. Then, the candidate sub-mesh with the maximum BV is stored. After all Na allocated sub-systems are identified, the candidate sub-mesh with maximum BV is selected. This BL allocation process takes O(Na^3). Subpartitions(or Buddies): 1) TL (top left), 2) BL (bottom left), 3) TR (top right), or 4) BR (bottom right) and assigned one (of size r X c or c X r) at level k+1 to the request, this Figure illustrates an example of the Q-Tree and the status of a system that stores 3 tasks (4x4, 2x3, and, 2x4) on a given 8x10 system.

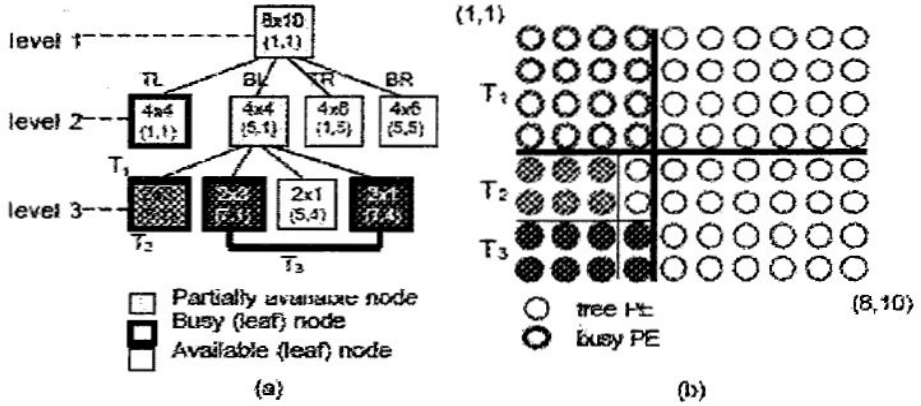

Figure 6: (a) A Q-Tree system state representation and (b) the corresponding 8x10 mesh system with 3 tasks allocated.

[Mustafa, 4(1): January 2017] ISSN 2348 – 8034
DOI- 10.5281/zenodo.263984 Impact Factor- 4.022

X. THE QUICK ALLOCATION STRATEGY

In 1997, the best fit QA (Quick Allocation) strategy was proposed in order to improve time complexity. In that strategy, the following data structures were used: (1) a busy sub-system list (of allocated Na tasks), (2) a coverage sub-system list, and (3) reject areas. For an incoming task rXc, the searching process was to find an available sub-system (it began by computing the coverage subsystem list and the reject areas). Then, all coverage sub-systems were sorted in non-decreasing order. For each row (starting from 1 to R of N = R x C), find a free sub-system (that did not intersect with the coverage sub-systems and the rejected areas) and allocate it to the request. The time complexity of the QA allocation is O(Na√N), and the performance improved over the AS strategy.

XI. CONCLUSIONS

In this research we Introduced a new more efficient best fit Q-Treebased sub-system allocation. New and more powerful best-fit criteria are used (i.e., the "maintainmaximum free sizes" criterion, the "minimum different size factor" criterion, the "maximum free size after partitioning" criterion, and the "minimum combining factor" criterion), and the time complexity has been reduced to 0(Na), where Na is the number of allocated tasks (Na<N) and N is the system size. By simulation studies, a number of experiments were carried out to investigate and evaluate the system performance when applying the Q-Tree strategy and compare it to other existing strategies. System performance was measured in terms of system utilization, system fragmentation, and average task completion time. The simulation results showed that the Q-Tree approach yielded the best system utilization and the best average task completion time, representing an improvement of up to 33% over BL, FSL, and QA approaches.

REFERENCES

The Web Sites and the works referred are:

1. *best fit of memory management.*
2. *best fit allocation.*
3. *www.dcc.ufla.br/-heitor/Projetos.html*
4. *CS431: Introduction to Operating Systems/memory management/ Vijay Kumar*
5. *Principle of memory management/ chapter 9*
6. *A New "Quad-Tree-Based" Sub-System ALLocation Technique for Mesh-connected Parallel Machines.*
7. *-JeerapornSrisawat and Nikitas A. Alexandridis Department of Electrical Engineering and Computer Science The George Washington University Washington DC 20052, U.S.A. {jeera, alexan}@seas.gwu.edu.*
8. *Underestanding operating system/sixth adition/chapter 2/memory management.*
9. *Alverson, R. et al. The Tera computer System. Proc. 1990 Int'l Conf. Supercomputing, 1990, 1-6.*
10. *Chuang, P.J. and N.F. Tzeng, An Efficient Submesh Allocation Strategy for Mesh Computer Systems. Proc. Int'l Conf. on Distributed Computing Systems, May 1991, 256-263.*
11. *Chuang, P.J. and N.F. Tzeng, Allocating Precise Submesh in*
12. *Mesh-Connected Systems. IEEE Trans, on Parallel and Distributed Systems, v.5(2), 1994, 211-217.*