

Short Paper

Allocation-Time-Based Processor Allocation Scheme for 2D Mesh Architecture*

XIAOMEI ZHU AND WEI-MING LIN⁺

Personal System Group

Dell Computer Corp.

Austin, TX 78758-4055, U.S.A.

E-mail: Xiaomei_Zhu@Dell.Com

⁺*Division of Engineering*

The University of Texas at San Antonio

San Antonio, TX 78249, U.S.A.

E-mail: wlin@voyager1.utsa.edu

Mesh is a widely used architecture in parallel computing systems. Research on efficient allocation of processors to incoming tasks on mesh architecture is very important in achieving the desired high performance. The processor allocation strategy proposed in this paper is based on a well-known boundary search approach and considers allocation time as a primary allocation decision-making factor. In this proposed technique, an additional novel heuristic is employed to consider, whenever feasible, having adjoining submeshes allocated to tasks with similar allocation times. The external fragmentation problem is expected to be alleviated, leading to improvement in utilization and shorter task waiting time. Another novel add-on feature is also employed to provide extra improvement. Our simulation results demonstrate substantial improvement in performance.

Keywords: parallel processing, processor allocation, mesh architecture, processor utilization, task waiting delay

1. INTRODUCTION

In parallel computing systems, efficient allocation of processors to incoming tasks is necessary to achieve the desired high performance. An efficient allocation algorithm should run fast, have low memory cost and result in short average task waiting time and high system resource utilization. In the literature, there already are many different processor allocation strategies for systems with various connection topologies. Performance as well as overhead varies depending on whether processors allocated for a task have to be contiguous or can be separate, with the latter case having better potential for processor utilization while posing a

Received March 26, 1999; revised July 5, 1999; accepted September 8, 1999.

Communicated by Chyi-Nan Chen.

*This research was supported in part by the Office of Naval Research under grants N00014-95-1-0514 and N00014-96-1-0897, and in part by the Department of Defense/Air Force Office of Scientific Research under grant F49620-96-1-0472

problem of higher intra-task communication overhead. In this paper, we are only concerned with processor allocation techniques employed on a two-dimensional mesh-connected architecture with contiguous allocation. A number of allocation schemes have been proposed in this area of research, such as the 2D Buddy strategy of Li and Cheng [6], the Frame Sliding (FS) scheme of Chuang and Tzeng [2], the First Fit and the Best Fit strategies of Zhu [9], the Adaptive Scan strategy of Ding and Bhuyan [3], the boundary search method of Sharma and Pradhan [7] and the Quick Allocation strategy of Seong-Moo Yoo [8].

The 2D Buddy strategy allocates to incoming tasks only square submeshes with their dimensions limited to powers of two. This strategy is simple and does not lead to any external fragments; however, it could cause serious internal fragmentation by overallocating tasks not requiring submeshes with such a dimension. The Frame Sliding scheme solves the overallocation problem by allocating a free submesh (a frame) which exactly matches the need of the incoming task. It uses a special method to slide the frame in order to locate an available submesh. Although the search process is not time-consuming, it results in external fragments and the problem of “miss allocation.” The First Fit (FF) method scans through available base nodes and stops the scanning process once a satisfactory free submesh is found. This method is fast but often leads to poor allocation. The Best Fit (BF) method searches for all free regions that are large enough to meet the task requirements and chooses the one with the most busy neighbors. Although BF does seem to be beneficial, the inadequacy of its associated heuristics renders its performance far from optimal, and it suffers from high computation overhead. The Adaptive Scan (AS) scheme scans through all the free regions to allocate a submesh to an allocation request in both dimensions, an improvement over the FS scheme. The boundary search method allocates a free submesh with the largest number of adjacent busy nodes to an allocation request by checking through all the candidate submeshes. The heuristic is based on the notion that if as many busy submeshes as possible are adjoined to one another, then less external fragmentation will result. The Quick Allocation strategy tries to reduce the allocation time and waiting delay by using an efficient information collecting procedure and a fast algorithm to identify base nodes.

Among all the above research on producing a fast allocating algorithm and at the same time avoiding internal as well as external fragmentation, the boundary search scheme has proved to be a promising approach because its performance is better than that of all the existing approaches. However, when submeshes are deallocated with an undesirable sequence and timing, heuristics in this scheme may not provide the best allocation. In this paper, we believe that if a task is allocated to a submesh that is adjacent to others which have *similar* deallocation times, less external fragmentation will result. Although in most practical parallel environments, deallocation times are not available for this purpose, depending on the variance of the job’s required service time, allocation times can be used as an alternative in measuring this *similarity* of deallocation times with a certain degree of reliability. By combining the two heuristics, i.e. the maximum boundary value with the weighted allocation time difference, the proposed approach can easily outperform the boundary search scheme without adding extra complexity. We also present an add-on feature, a simple tie-breaking technique, for the boundary search scheme to obtain further improvement. This is extremely beneficial when multiple best choices occur frequently from the original search method.

The rest of this paper is organized as follows. All preliminaries, including the problem definition and underlying method, are described in section 2. Our proposed strategy is then presented in section 3, followed by simulation results in section 4. Concluding remarks are given in section 5.

2. PRELIMINARIES

2.1 2D-Mesh Architecture

The mesh-connected multiprocessor system is one of the most widely adopted parallel architectures. The two-dimensional mesh architecture has been one of the most popular architectures because of its scalability and simplicity. The mesh topology is widely adopted in many parallel computer systems, among which the most notable are the Intel Touchstone [4], the Intel Paragon XP/S [5], and the Tera Computer Systems [1].

2.2 Problem Definition

Processor allocation is a problem in which an incoming task is to be allocated using some processors in a parallel system. From the perspective of the system, the processors can be time-shared or space-shared. In time-shared allocation, processors are shared by tasks by dividing the time into separate slots, with each slot allocated to a distinct task. On the other hand, space-shared allocation divides the whole system into physical partitions of processors and then allocates partitions to incoming tasks without any sharing of any processor among different tasks. In the literature, most of the research works have focused on the space-shared processor allocation methods. Space-shared methods can be further divided into two categories, contiguous and non-contiguous. In a contiguous method, all partitions are connected graphs while separate regions of processors can be allocated to a task in a noncontiguous method. It is well known that, although a noncontiguous method may result in less external fragmentation, communication overhead incurred for communication-intensive tasks may easily offset any performance benefit produced. The goal of this paper is to exploit a novel heuristic to produce a better contiguous processor allocation technique for a two-dimensional mesh architecture, where each incoming task requests a submesh rectangular in shape.

2.3 Performance Metrics

There are several metrics that have been used to evaluate the performance of a processor allocation technique. *The average task waiting time* corresponds to the mean time a task spends waiting for being served. *The average response time* measures the mean time from when a task enters the system to when it finishes. Another metric is the *system throughput*, which gives the rate at which the system provides service to tasks.

2.4 Boundary Search Method (BSM)

As noted above, the allocation technique employed in this paper is the Boundary Search Method (*BSM*) presented in [7]. A few critical definitions and fundamental concepts are given here for the sake of completeness.

A two-dimensional mesh $M(L, W)$ consists of $N = L \times W$ nodes (processors) arranged in a two-dimensional grid of length L and width W . Terms used throughout the discussion are defined in the following.

- A *submesh* $S(l, w)$ in the mesh $M(L, W)$ is a rectangular grid of nodes belonging to $M(L, W)$ with length l and width w ($l \leq L$ and $w \leq W$).

- A *free submesh* is a submesh in which all nodes are free. An *allocated submesh* is a submesh currently allocated to a task.
- The *boundary value* of a free node in the mesh $M(L, W)$ is the total number of allocated neighbors of this node plus the number of mesh boundary points on which it lies.
- The *boundary value* of a free submesh M is the sum of the boundary values of all its nodes in the periphery. In other words, if S is the set of all allocated submeshes that are neighbors of M , then the boundary value of M , denoted as $BV(M)$, can be represented as

$$BV(M) = \sum_{m \in S} bv(M, m),$$

where $bv(M, m)$ denotes the degree of adjacency (i.e., the number of adjacent nodes) between M and m . Note that the four mesh boundary sides are always elements in S .

Fig. 1 shows an example of boundary values for two candidate submeshes. Note that the boundary value of the top-leftmost node in the left candidate submesh is two according to the definition.

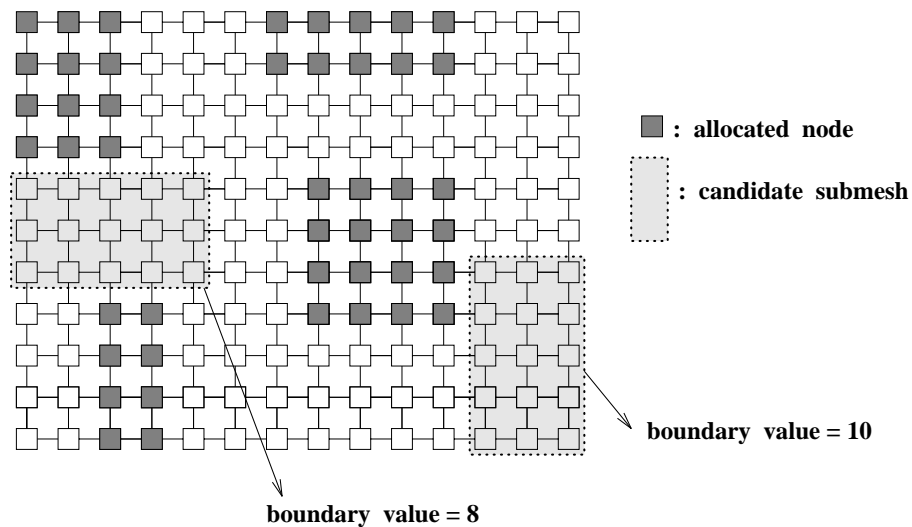


Fig. 1. An example of boundary values.

The *BSM* will search for the candidate submesh that has the maximum boundary value for allocation. With this criteria, the search process for candidate submeshes becomes very efficient because it skips over spaces which lead to zero boundary values for any potential candidates. Simulation results [7] indicate that *BSM* outperforms all existing strategies in terms of the average waiting time and variance, with a very respectable allocation time complexity of $O(N_A^3)$, where N_A is the number of allocated submeshes in the whole mesh.

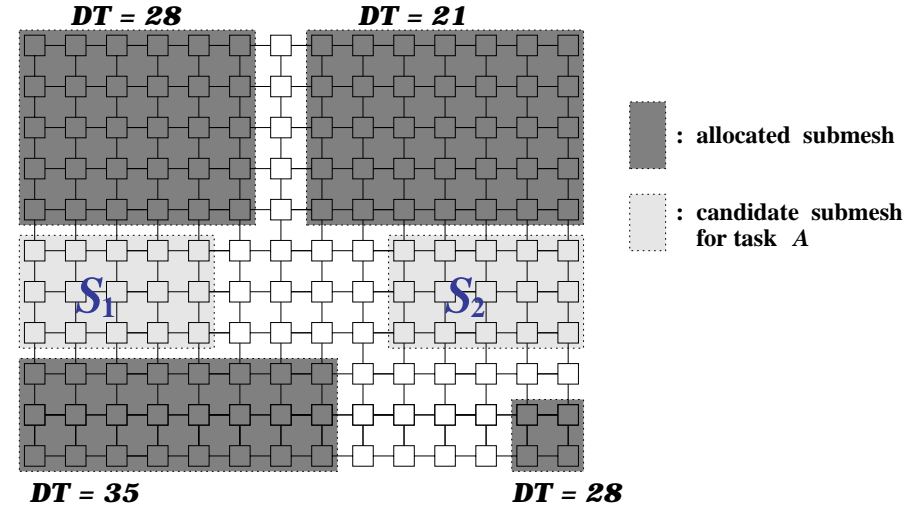
3. PROPOSED PROCESSOR ALLOCATION SCHEMES

3.1 Idea and Definition

The heuristic adopted by *BSM* allocates a candidate submesh with the maximum boundary value to an allocation request. In most cases, the more “adjacencies” a new allocation has with respect to allocated submeshes, the less external fragmentation will result. However, if such “adjacencies” are located between submeshes that have their deallocation times apart, the fragmentation problem may seem less severe for a shorter period of time, but the problem may last longer. Such long lasting fragmentation will hinder processor utilization by preventing a larger free space from forming in time for large size requests.

Although, in most practical parallel environments, deallocation times are not available for this purpose, depending on the variance of a job’s required service time, allocation times can be used as an alternative in measuring this *similarity* among deallocation times with a certain degree of reliability. That is, two jobs with similar allocation times tend to have similar deallocation times with a high probability than the two otherwise. The fundamental concept behind our approach is to not only utilize the “boundary value” heuristic in *BSM*, but also to attenuate/augment it according to the difference in the allocation times of the adjacent submeshes. It is not difficult to see that when two adjacent submeshes are to be deallocated close to each other time-wise, the fragmentation scenario after the first one becomes deallocated will last for a shorter time. Now the question becomes: “For a submesh request, is it better to allocate a submesh with the largest boundary value which will lead to the least fragmentation right after this allocation, or to allocate one that will lead to a **larger** free submesh when this submesh gets deallocated along with its neighbors in the future?” Fig. 2 shows an example with four allocated submeshes and specified deallocation times (*DT*). Here it is assumed that the processing time for task allocation is minimal compared to a time unit displayed for measurement. When searching for candidate submeshes for Task *A*, the next task in the queue, the candidate submesh on the left (S_1) will give the best boundary value of 13 while the one on the right (S_2) will result in a boundary value of 8. If a decision is made solely based on this criterion, then task *A* will be allocated with S_1 , and task *B* will in turn be allocated with S_2 , which will subsequently block the allocation for task *C* until time equals 28 when the submesh in the upper-left becomes free. If the *DT* similarity is “somehow” taken into consideration, and if task *A*, which has a *DT* of 20 is allocated instead with S_2 , which has a neighbor with a similar *DT* (21), and if task *B* is assigned to S_1 , then task *C* will be allocated at a much earlier time, 21 (versus 28), when both task *A* and its neighbor are deallocated.

There are many ways to incorporate these two heuristics together. As mentioned above, the *DT* difference is replaced by the allocation time difference (*ATD*) when the heuristic is applied. In this paper, a very simple weighting system is used to augment or attenuate the boundary value between every two adjacent submeshes originally calculated in *BSM*. This argumentation or attenuation process is by multiplying a coefficient depending on the *ATD* between the two adjoining submeshes. The same *BSM* can then be applied by using this newly calculated weighted boundary value for submesh selection. Our proposed allocation algorithm is thus called the Weighted Boundary Value Method (*WBSM*).



task in queue	A	B	C
submesh size requested	3x5	3x5	7x8
projected execution time	10	20	25

current time = 10

Fig. 2. An example showing an insufficient heuristic based on boundary values.

3.2 Weighting Coefficient Function

A weighting coefficient (*WC*) function of *ATD* has to be derived for the proposed *WBSM*. All *WC* values are chosen so as to be between 0 and 1. Thus, naturally, a boundary value with a larger *ATD* should be attenuated with a smaller coefficient, and a typical *WC* function should be in the form of the function depicted in Fig. 3. For the sake of simplicity for implementation, the actual *WC* function employed is chosen so as to be a linear function as shown in Fig. 4. As *ATD* becomes large, *WC* should tail off quickly to reflect the irrelevance of two tasks having large temporal discrepancy in their allocating times. The cut-off point in the selected linear function shown in Fig. 4, α , remains to be determined.

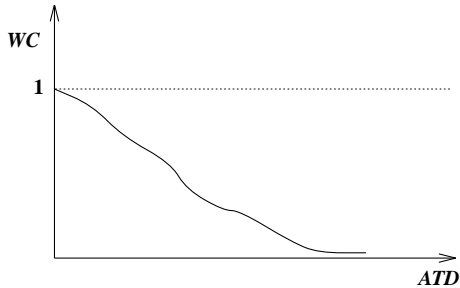


Fig. 3. A typical weighting coefficient function.

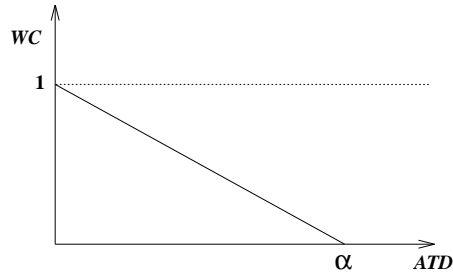


Fig. 4. Weighting coefficient function used for simulation.

Note that, since the four boundary sides of the mesh will always exist, it is always more beneficial to allocate submeshes adjacent to them. Thus, a WC of 1 will be used for all boundary values incurred due to these boundary sides. To apply this WC function, each “boundary point” originally tallied with a value of 1 in BSM is now weighted by multiplying it by the WC value according to the ATD between the two adjoining submeshes contributing to this boundary point. That is, the new Weighted Boundary Value (WBV) of a free submesh M becomes

$$WBV(M) = \sum_{m \in S} WC(ATD(M, m)) \times bv(M, m),$$

where $ATD(M, m)$ and $bv(M, m)$ indicate the allocation time difference and boundary value between M and a neighboring allocated submesh m , respectively.

3.3 A Simple Add-On Feature: Corner Value-Boundary Search Method (CVBSM)

In BSM , the candidate submesh with the maximum boundary value is chosen for allocation. When there are several candidates with the same maximum boundary value, BSM chooses the first one found. Close observation shows that in most situations, there exist many such candidates with the same maximum boundary value. Thus, the one selected by BSM usually does not lead to the best allocation result.

Note that, in general, if a candidate submesh with more corners surrounded by the allocated area is selected for allocation, then less external fragmentation results. The *Corner Value* of a corner of a candidate submesh is the number of busy nodes in the three nodes surrounding the corresponding corner point, i.e., the number of busy nodes among the three nodes in the horizontal, vertical and diagonal directions. The *Corner Value* of a candidate submesh is the sum of all the *corner values* of its four corners. For example, the *Corner Value* of the candidate submesh S_1 shown in Fig. 2 is 10.

As mentioned above, the proposed $CVBSM$ is based on BSM . The candidate submesh for allocation is the one with the maximum corner value among all the candidates having the same maximum boundary value. All the corner values can be easily obtained when calculating the boundary values of candidate submeshes according to their positions relative to the candidate submesh and the allocated submeshes. Therefore, very minimal extra overhead time is added when this technique is used.

4. SIMULATION RESULTS

The performance of the proposed $WBSM$ and $CVBSM$ is compared here with that of the original BSM based on a series of simulation runs. The mesh in our simulation was of size $M \times N$. A few assumptions and simulation parameters are described as follows:

- The dimensions of the submesh $m \times n$ requested by an incoming task were generated such that m was uniformly distributed over $(1, M_d)$ and n was uniformly distributed over $(1, N_d)$, where $1 \leq M_d \leq M$ and $1 \leq N_d \leq N$.
- The service time of incoming tasks was normally distributed with a mean of μ and a standard deviation of σ .

- The inter-task arrival time was exponentially distributed with an average interarrival time of $1/\lambda$.
- 200,000 submesh requests (tasks) were simulated in each simulation run to ensure that an equilibrium state was reached, if possible. This was needed to determine whether the task interarrival time ($1/\lambda$) was too small for the task queue to remain in an equilibrium state.
- The cut-off point in the selected linear weighting coefficient function shown in Fig. 4, α , was set to μ since an *ATD* value much larger than μ would have minimal impact on the boundary value weighting scheme.

The first group of simulation runs was used to determine the relationship between the average task waiting time and the average inter-task arrival time ($1/\lambda$), with $M = M_d = 200$, $N = N_d = 200$, $\mu = 100$ and $\sigma = 25$. The simulation results of *WBSM*, *CVBSM*, and *BSM* are displayed in Fig. 5 while the improvement obtained by the two proposed methods versus *BSM* is shown in Fig. 6.

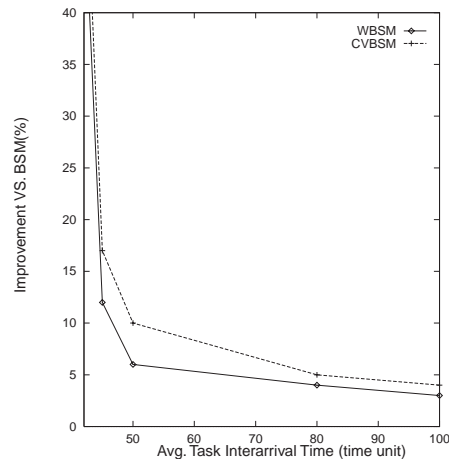
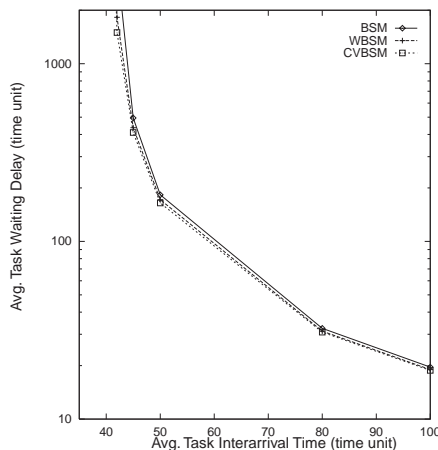


Fig. 5. Average Waiting Delay vs. $1/\lambda$ when $M = M_d = 200$, $N = N_d = 200$, $\mu = 100$ and $\sigma = 25$.
 Fig. 6. Improvement in the Average Waiting Delay vs. $1/\lambda$ when $M = M_d = 200$, $N = N_d = 200$, $\mu = 100$ and $\sigma = 25$.

From the results, we can see that when the load is light, *WBSM* and *CVBSM* exhibit 5% to 10% improvement over *BSM* and much more pronounced improvement when the load becomes heavier.

The second group of simulation runs was used to determine how *WBSM* and *CVBSM* compare to *BSM* when the dimensions of the mesh vary. The parameters used where: $M = M_d = N = N_d$, $\mu = 100$, $\sigma = 25$ and $1/\lambda = 45$. Result are shown in Fig. 7. We can see that *WBSM* and *CVBSM* maintain steady improvement over *BSM* when the mesh size varies. Furthermore, *WBSM* exhibits much greater improvement of 30% to 40%.

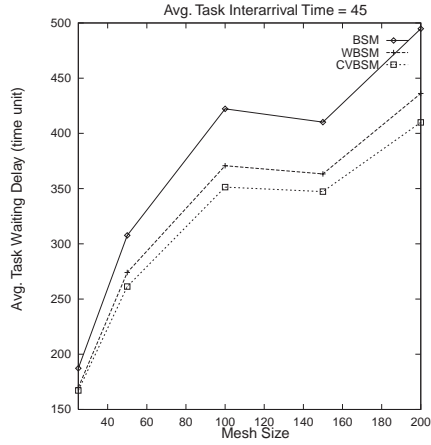


Fig. 7. Average Waiting Delay vs. $M(=N)$ when $M = M_d$, $N = N_d$, $\mu = 100$, $\sigma = 25$ and $1/\lambda = 45$.

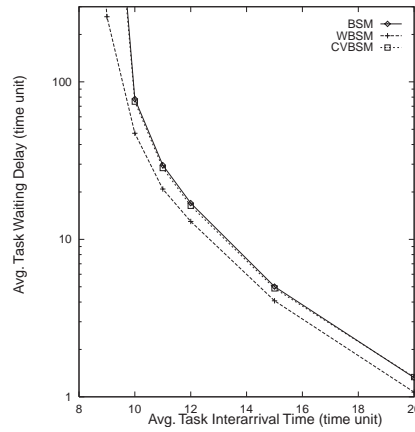


Fig. 8. Average Waiting Delay vs. $1/\lambda$ when $M = N = 200$, $M_d = N_d = 100$, $\mu = 100$ and $\sigma = 25$.

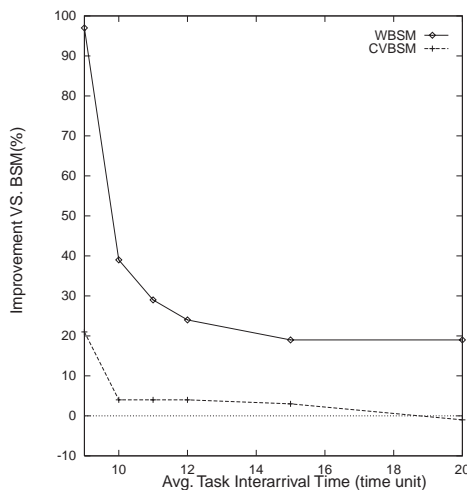


Fig. 9. Improvement in Average Waiting Delay vs. $1/\lambda$ when $M = N = 200$, $M_d = N_d = 100$, $\mu = 100$ and $\sigma = 25$.

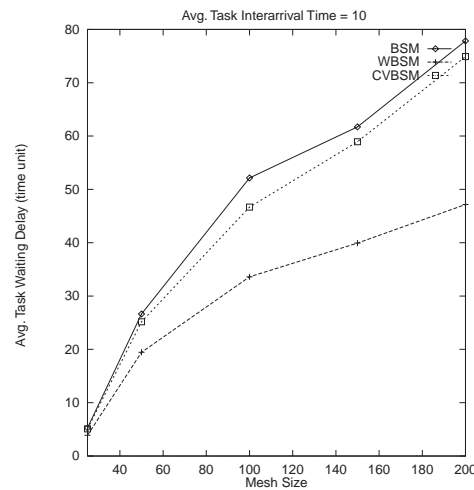


Fig. 10. Average Waiting Delay vs. $M(N)$ when $M_d = M/2$, $N_d = N/2$, $\mu = 100$, $\sigma = 25$ and $1/\lambda = 10$.

While limiting the requested submesh sizes to a smaller range relative to the mesh size, another simulation run was conducted with $M_d = M/2 = 100$ and $N_d = N/2 = 100$. Corresponding results are shown in Fig. 8, Fig. 9 and Fig. 10 respectively. Under this setup, *WBSM* exhibited even better results while there was no further improvement for *CVBSM*. This was due to the fact that more jobs that could be accommodated in the mesh simultaneously in this case. *WBSM* is capable of making better selections in this more complicated situation. On the other hand, the probability that candidate submeshes will have the same maximum boundary value tends to be smaller, which results in insignificant improvement results for *CVBSM*.

5. CONCLUSIONS

In this paper, we have proposed an improved method based on an already prominent boundary search approach which considers the difference in allocation times as another parameter. Compared to the original method, the proposed method has the same complexity in allocation time and deallocation time and requires no extra memory space. Simulation results indicate significant improvement in the average waiting time as well as the system throughput. An interesting and challenging goal in future work will be identified an optimal WC function for this proposed approach.

REFERENCES

1. Alverson et al., "The tera computing system," in *Proceedings of 1990 International Conference on Supercomputing*, 1990, pp. 1-6.
2. P. J. Chuang and N. F. Tzeng, "An efficient submesh allocation strategy for mesh computer systems," in *Proceedings of International Conference on Distributed Computing Systems*, 1991, pp. 256-263.
3. J. Ding and L. N. Bhuyan, "An adaptive submesh allocation strategy for two-dimensional mesh connected systems," in *Proceedings of International Conference on Parallel Processing*, 1993, pp. II-192-200.
4. *A Touchstone DELTA System Description*, Intel Corporation, 1991.
5. *Paragon XP/S Product Overview*, Intel Corporation, 1991.
6. K. Li and K. H. Cheng, "A two-dimensional buddy system for dynamic resource allocation in a partitionable mesh connected system," *Journal of Parallel and Distributed Computing*, Vol. 12, No. 1, 1991, pp.79-83.
7. D. D. Sharma and D. K. Pradhan, "A fast and efficient strategy for submesh allocation in mesh-connected parallel computers," in *Proceedings of Symposium on Parallel and Distributed Processing*, 1993, pp. 682-689.
8. S. M. Yoo, et al., "An efficient task allocation scheme for 2D mesh architectures," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 9, 1997, pp. 934-942.
9. Y. Zhu, "Efficient processor allocation strategies for mesh-connected parallel computers," *Journal of Parallel and Distributed Computing*, Vol. 16, No. 4, 1992, pp. 328-337.

Xiaomei Zhu (朱晓梅) received her BS degree in Computer Science from Xi'an Jiaotong University, China, in 1991, and MS degree also in Computer Science from Huazhong University of Science and Technology, China, in 1994. She then went to the University of Texas at San Antonio in 1997 and received the MS degree in Electrical Engineering in 1998. Since then, she has been with Dell Computer Corporation in Austin, Texas. Her research interests are mainly in the areas of parallel and distributed systems.

Wei-Ming Lin (林維明) received the BS degree in Electrical Engineering from National Taiwan University, Taipei, Taiwan, in 1982, and the MS and Ph.D degrees in Electrical Engineering from the University of Southern California, Los Angeles, in 1986 and 1991, respectively. He was an assistant professor in the Department of Electrical and Computer Engineering at Mississippi State University before joining the University of Texas at San Antonio in 1993, and, since 1998, he has been an associate professor of Electrical Engineering there. Dr. Lin has published more than 50 papers in international journals and conferences in the area of distributed and parallel computing, and computer architecture.

