이차원 패킹 알고리즘의 이론적 성능 분석과 병렬화

황 인 재[†] · 홍 동 권[††]

요 약

이차원 패킹 알고리즘은 메쉬 멀티프로세서 시스템을 분할 및 할당하는데 유용하게 활용될 수 있다. 기존연구에서 TP 휴리스틱 알고리즘이라 불리는 효율적인 패킹 알고리즘을 개발하였으며 패킹 결과가 어떻게 메쉬 멀티프로세서 시스템을 분할 및 할당하는데 활용될 수 있는지 보여주었다. 본 논문에서는 TP 휴리스틱 알고리즘의 이론적인 성능분석결과를 제시한다. 또한 알고리즘을 병렬화하여 다수의 프로세서를 이용하여 수행되었을 때 보다 적은 수행시간을 소모하게 한다.

Theoretical Performance Bounds and Parallelization of a Two-Dimensional Packing Algorithm

Injae Hwang[†] · Dong-Kweon Hong[††]

ABSTRACT

Two-dimensional packing algorithm can be used for allocating submeshes in mesh multiprocessor systems. Previously, we developed an efficient packing algorithm called TP heuristic, and showed how the results of the packing could be used for allocating submeshes. In this paper, we present theoretical performance bounds for TP heuristic. We also present a parallel version of the algorithm that consumes reduced time when it is executed by multiple processors in mesh multiprocessors.

키워드 : 메쉬형 병렬컴퓨터(Mesh Multiprocessors), 이차원 패킹(Two-dimensional Packing), 이론적 성능한계(Theoretical Performance Bound), 병렬알고리즘(Parallel Algorithm), 프로세서 할당(Processor Allocation)

1. Introduction

Two-dimensional packing problem has been studied by many researchers [1-4]. It arises in a variety of situations such as scheduling of tasks and cutting-stock problems. Cutting-stock problems may involve cutting objects out of a sheet or roll of material so as to minimize waste. The scheduling of tasks with a shared resource involves two dimensions, the resource and time, and the problem is to schedule the tasks so as to minimize the total amount of time used. In general, the problem is stated as follows : Given a rectangular bin with fixed width and infinite height, pack a finite set of rectangles of specified dimensions into the bin in such a way that the rectangles do not overlap and the total bin height used in the packing is minimized.

We showed that two-dimensional packing could also be useful for allocating submeshes in mesh multiprocessor systems [4]. In the problem we studied previously, there are m tasks which have rectangular structures such as two-dimensional grids. These tasks can be executed independently on the submeshes allocated to them. Our problem is partitioning a given mesh multiprocessors into m submeshes in such a way that the workload is balanced and inter-processor communication is minimized. We adapted two-dimensional packing to solve such a processor allocation problem. We developed an efficient heuristic packing algorithm called TP(tight-pack) heuristic, and showed how the result of packing could be used for partitioning a given mesh.

In this paper, we present theoretical performance bounds for TP heuristic. Even though experimental results showed that the heuristic packing algorithm performs well for a variety of cases, the bounds presented here guarantee that it will never produce solution values that exceed a certain limit. We also present a parallel version of the algorithm that consumes reduced time when it is executed by multiple processors in mesh multiprocessors.

The organization of this paper is as follows. In the next

† 종신회원 : 충북대학교 컴퓨터교육과, 컴퓨터정보통신연구소 교수
†† 종신회원 : 계명대학교 컴퓨터 · 전자공학부 교수
논문접수 : 2002년 8월 14일, 심사완료 : 2002년 12월 31일

section, we survey some of the related works to this paper. In section 3, we briefly explain TP algorithm and show how the results of packing can be used to solve the submesh allocation problem. In section 4, we prove that the heuristic algorithm is guaranteed to produce solution values that do not exceed a certain limit. In section 5, we present parallel version of the algorithm that can be executed efficiently on mesh multiprocessors. Finally, we give the summary of the paper in section 6.

2. Related Work

Two-dimensional packing has been used for solving scheduling of tasks and cutting-stock problems. In this section, we survey some of the recently published works that are related to this paper.

Azar and Epstein considered packing of rectangles into an infinite bin [1]. Similar to the Tetris game, the rectangles arrive from the top and, once placed, cannot be moved again. The rectangles are moved inside the bin to reach their place. For the case in which rotations are allowed, they designed an algorithm whose performance ratio was constant. In contrast, if rotations are not allowed, they showed that no algorithm of constant ratio exists. For this case they designed an algorithm with performance ratio of $O(\log\frac{1}{\varepsilon})$, where ε is the minimum width of any rectangle.

Hifi and Ouafi discussed the problem of packing a set of small rectangles (pieces) in an enclosing final rectangle [3]. They presented first a best-first branch-and-bound exact algorithm and second a heuristic approach in order to solve exactly and approximately this problem. The performances of the proposed approaches were evaluated on several randomly generated problem instances. Computational results show that the proposed exact algorithm is able to solve small and medium problem instances within reasonable execution time.

Paulhus also presented an algorithm that can be used to pack sets of squares (or rectangles) into rectangles [6]. The algorithm was applied to three open problems and showed how the best known results could be improved significantly.

3. A Heuristic Packing Algorithm

In many applications, a task can be represented by a two-dimensional grid. In the formulation of our processor allocation problem, we assume that m rectangular grids are given as independent tasks. The number of processors N in the given mesh multiprocessors is assumed to be larger than m, so that m disjoint submeshes can be allocated to the grids. Each grid point of a grid represents a certain amount of computation, hence its computational workload is proportional to the number of grid points. The computation on a grid point (except the ones on boundaries) need data from its four neighbors. If a grid is assigned to a set of processors (a submesh), the communication cost between two processors is proportional to the number of grid points assigned to a processor whose neighbors are assigned to another processor. Assume that $w\times h$ grid is assigned to $X\times Y$ processor submesh. The grid is uniformly divided into XY pieces with dimension $\frac{w}{X}\times\frac{h}{Y}$, so that one can be assigned to each processor. Then, the computational cost is proportional to $\frac{wh}{XY}$ which is the average number of grid points assigned to each processor. Communication cost is proportional to $2(\frac{w}{X}+\frac{h}{Y})$ which is the number of grid points on boundaries of a piece of grid assigned to a processor.

To perform the computation of the grids on mesh multiprocessors, it is necessary to find m submeshes and their locations, one for each grid. In the allocation strategy we proposed, we first pack the given set of grids using the ratio of processor mesh $R=\frac{P}{Q}$ (we are given a processor mesh $P\times Q$) [4]. The grids are packed in such a way that the ratio of width to height of the space used for packing grids is as close to R as possible. Then we use the two ratios $\frac{width}{P}$, $\frac{height}{Q}$ to allocate a submesh to each grid.

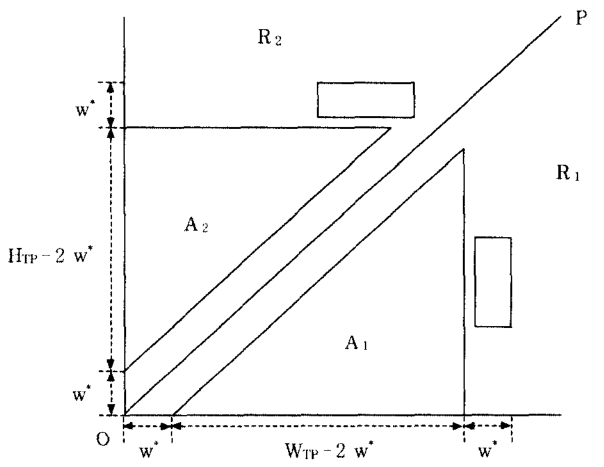
The basic idea of TP-heuristic is as follows. First the grids are sorted in some selected order. Then we start packing grids one by one at the south-west corner of the bin. (The width of the bin is assumed to be infinite) Let's consider the space of the bin as the first quadrant of $X-Y$ plane. Then the south-west corner of the bin becomes the origin of the coordinate system, that is (0, 0). Each packed grid has 4 corners **NW, NE, SE** and **SW** with respect to its orientation in the packing. A **NW** or **SE** corner of a packed grid is called a **free corner (FC)** if no other item occupies that corner. In our algorithm, only free corners are considered for packing the new grid. When a new grid is placed in a free corner, it is placed so that it is above and to the right of the corner. After packing the first grid at the origin, the next grid is packed at one of the two corners created by packing the first grid. We also keep the maximum size of the grid which can be packed at the free corner (we call it the size of free corner) along with its location. We choose a free corner for the $i+1$-st grid, so that the maximum of W_{i+1} and H_{i+1} is minimized. Assuming that we are given a processor mesh $P \times Q$ with $P \geq Q$ and $\frac{P}{Q} = R$, we choose the corner for the $i+1$-st grid, so that the maximum of W_{i+1} and RH_{i+1} is minimized. Since the number of free corners cannot be larger than $m+1$ at any time, the time complexity of the algorithm is $O(m^2)$.

4. Theoretical Performance Bounds

To show a bound for the accuracy of the solutions provided by our packing algorithm, we impose the following restrictions on packing the grids. When $w_i \times h_i (w_i \geq h_i)$ grid is packed at a corner (x, y), it should be placed so that the side with dimension w_i (long side) is parallel to the X-axis if $x < Ry$ and it should be placed with long side parallel to the -axis if $x > Ry$. Suppose there is a free corner that cannot accommodate the item in the allowed orientation but can accommodate the item in the other orientation. Then we disregard this corner though it is possible to get a better packing by placing the item in that corner. If $x = Ry$, then both orientations of the grid are allowed for that corner. Now we state a result on the solution accuracy bound when $R=1$ (i.e. for square meshes)

Theorem 1 : Consider the two-dimensional packing problem with $R=1$ and let $w^* = \max_i w_i$. Let W_{opt}, H_{opt} be the width and height of the optimal packing and let W_{TP}, H_{TP} be the corresponding values for the packing given by the TP-heuristic when items are packed in decreasing order of their maximum side lengths. Assume without loss of generality that $W_{opt} \geq H_{opt}$ and $W_{TP} \geq H_{TP}$. Then $W_{TP} \geq \sqrt{2}\, W_{opt} + 3w^*$.

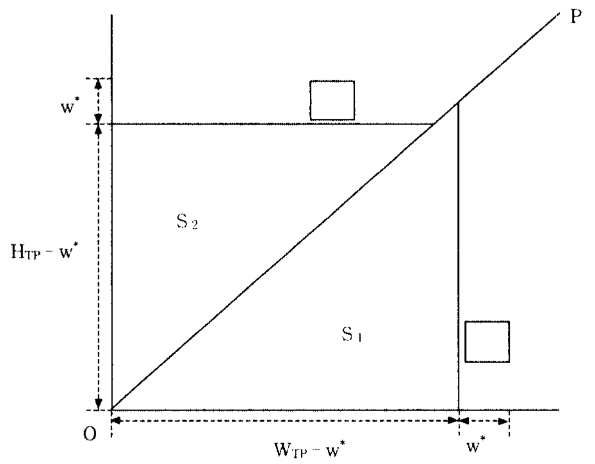
Proof : Let us denote the regions below and above the line OP (that has unit slope) by R_1 and R_2. First we observe that there is always a corner in R_1 as well as in R_2 that can accommodate a subsequent item in the allowed orientation (i.e. long side parallel to Y-axis in R_1 and parallel to X-axis in R_2). This follows from the fact that we can always pack a subsequent item q abutting to Y-axis (or X-axis). Since the item q' below (or to the left of) q was packed prior to q, the maximum side of q' is longer than the maximum side of q. Hence, there is always enough space to pack item q above (or to the right of) item q'.



(Figure 1) Regions A_1 and A_2 in the packing (Theorem 1)

Now, we show that $W_{TP} - H_{TP} \leq w^*$ as follows. Let W_{TP}^i, H_{TP}^i be the width and height of the packing after the i-th item is packed. Suppose that $W_{TP} - H_{TP} \leq w^*$. Then there must be an item i of dimensions (w_i, h_i) such that $W_{TP}^{i-1} - - H_{TP}^{i-1} \leq w^*$ and $W_{TP}^i - H_{TP}^i > w^*$. It also must be true that the i-th item was packed at a corner in R_1, hence $W_{TP}^{i-1} < W_{TP}^i$ and $H_{TP}^{i-1} \leq H_{TP}^i$. Let (x, y) be the location of a corner in R_2 which can accommodate the i-th item. Then $x < y \leq H_{TP}^{i-1}$. Since $x + w_i < H_{TP}^{i-1} + w^* < W_{TP}^i$ and $y + h_i \leq H_{TP}^{i-1} + w^* < W_{TP}^i$, the maximum of the width and height of the packing would be smaller if the item i were packed at the corner at (x, y). Hence the item i should not have been packed at a corner in R_1. Since we always pack items so that the maximum of the width and height of the packing is minimized, we can conclude that $W_{TP} - H_{TP} \leq w^*$.

We only have to prove our result when $W_{TP} > 3w^*$ in which case $H_{TP} > 2w^*$. Consider the two isosceles right triangles A_1 and A_2(see Figure 1) in regions R_1 and R_2 with areas $\frac{(W_{TP} - 2w^*)^2}{2}$, $\frac{(H_{TP} - 2w^*)^2}{2}$ respectively. Note that all items in the region $A_1(A_2)$ have been packed with their long sides parallel to $Y(X)$ axis. Thus the items are packed in these regions as in bottom-up left-justified (BL for short) strategy of [2]. We can make the similar argument on the occupancy of A_1 and A_2 as in [2]. Note that any vertical (or horizontal) cut through A_1 (or A_2) can be partitioned into alternating segments corresponding to cuts through unoccupied and occupied areas. Using the fact that there are items to the right of A_1 and above A_2 and considering the order in which the items are packed, we can show that the sum of the occupied segments is at least the sum of the unoccupied segments. By integrating the lines over $W_{TP}-2w^*$ (or $H_{TP}-2w^*$), we can verify that A_1 (or A_2) is at least half full. This means that $W^2_{opt} \geq 1/4((H_{TP}-2w^*)^2+(W_{TP}-2w^*)^2) \geq \frac{(H_{TP}-2w^*)^2}{2}$ and the result follows from the fact that $W_{TP}-H_{TP} \leq w^*$.



(Figure 2) Regions S_1 and S_2 in the packing (Theorem 2)

The bound can be improved when the items to be packed are square shaped.

Theorem 2 : Consider the same 2D packing problem as in Theorem 1 except that the items are square shaped. If the items are packed in decreasing order of their sizes in the TP-heuristic, then $W_{TP} \leq \sqrt{2}\, W_{opt}+2w^*$.

Proof : The proof of this theorem is similar to the proof of the previous theorem. It can be easily shown that $W_{TP}-H_{TP} \leq w^*$. Since all the items are square shaped, they are packed as in BL strategy in the two isosceles right triangles S_1 and S_2 (see (Figure 2)) with areas $\frac{(W_{TP}-w^*)^2}{2}$, $\frac{(H_{TP}-w^*)^2}{2}$ respectively [2]. Using the fact that there are items to the right of S_1 and above S_2 and considering the order in which the items are packed, we can show that S_1 and S_2 are at least half-occupied.. This means that $W^2_{opt} \geq 1/4((H_{TP}-w^*)^2+(W_{TP}-w^*)^2) \geq \frac{(H_{TP}-w^*)^2}{2}$ and the result follows from the fact that $W_{TP}-H_{TP} \leq w^*$.

5. Parallelization of the Packing Algorithm

The packing algorithm described in the previous section is sequential. One processor has to collect all the information about the grids from the other processors and execute the packing algorithm in order to find the processor allocation. The result of allocation should be communicated to all the processors. In this section, we present a parallel algorithm in which m processors cooperatively execute the packing algorithm for m grids in order to speed up the algorithm. The same packing method that was used in the sequential algorithm will be used in the parallel algorithm described here. Assume that we are given the mesh ratio R and m grids, $w_1 \times h_1$, $w_2 \times h_2, \cdots, w_m \times h_m$. After the following algorithm terminates, global variables, $XLoc_i$ and $YLoc_i$, contain the location of the corner where grid i was packed. $Orient_i$ is set to 1 if grid i was rotated and it is set to 0 otherwise. The algorithms for common operations, such as broadcasting finding minimum value, can be found in, and will not be repeated in this paper [5].

Initially, each grid j is located at processor S_j. (If all the grids are at one processor, $S_i = S_j$ for all i and j.) Processor S_j produces a packet $\langle (w_j, h_j), S_j \rangle$ for grid j, where (w_j, h_j) is the dimension of grid j and S_j is its own processor index. These packets contain the necessary information that forms the input data to our packing algorithms. The detailed description of the algorithm is given below. The topology of multiprocessors on which the algorithm runs, is mesh.

```
Algorithm Parallel Packing
(a) Let PS be a set of $m$ processors forming a submesh or a sub-
    cube, that is $PS = p_{b_i} | 0 \leq i \leq m-1$. The processors which
    produced packets send them to the processors in PS (one pac-
    ket per processor) using procedure TokenPacking. The pro-
    cessors in PS will do the remaining steps of our parallel packing
    algorithm.
(b) Sort packets according to the packing order using a parallel sor-
    ting algorithm. After sorting, assume that processor $P_{b_i}$ is hol-
    ding packet $\langle (w_i, h_i), S_i \rangle$.
(c) Store the initial free corner $[(0,0),(\infty,\infty)]$ at processor $P_{b_0}$. Ea-
    ch processor $P_{b_i}$ set both width and height to 0. Now, pack the
    grids one by one by performing $m$ iterations where in the $i$-th
    iteration $(0 \leq i \leq m-1)$ call procedure GridPacking $(i)$.
```

```
(d) After step (c), each processor $P_{b_i}$ has ($XLoc_i$, $YLoc_i$), that is,
    the free corner where grid ($w_i, h_i$) was packed and the width
    and height of the packing. Each processor $P_{b_i}$ include the above
    information in its packet $\langle (w_i, h_i), S_i \rangle$ and send it to $S_i$ using
    procedure SendPacketToSource.
end Parallel Packing ;
```

```
Procedure TokenPacking ;
// Here a subset of processors $P_{j0}, P_{j1}, \cdots, P_{jl}$ with $j_0 < j_1 < \cdots < j_l$
has one packet each and it is desired to store the packet of $P_{jk}$
in $P_k$ for $t \le k \le (t+l) \bmod m$ for some $0 \le t \le N-1$. //
```

```
Procedure GridPacking ($i$) ;
1. Call procedure Broadcast ($b_i, \langle (w_i, h_i), S_i \rangle$).
2. Call parallel procedure FindBestCorner ($(w_i, h_i), k$) to nd the
   corner ($x_k, y_k$) where the grid ($w_i, h_i$) is to be packed, and its
   orientation.
3. Processor $P_{b_i}$ set $XLoc_i$ and $YLoc_i$ to $x_k$ and $y_k$ respectively,
   and set $Orient_i$ to $Orient_k$. Also $P_{b_i}$ determines the width and
   height of the packing after ($w_i, h_i$) is packed.
Call procedure Broadcast ($b_i, ((x_k, y_k), (width, height), Orient_i)$).
4. Call parallel procedure UpdateCorners ($(w_i, h_i), k$) to update
   the sizes of corners after the grid ($w_i, h_i$) is packed at ($x_k, y_k$).
5. Call Parallel procedure FindNewCornerSize ($(w_i, h_i), k$) to
   determine the sizes of two new corners.
```

```
Procedure FindBestCorner ($(w_i, h_i), k$) ;
1. Each Processor $P_{b_j}$ does the following
   begin
         Let [$(x_j, y_j), (p_j, q_j)$] the corner $P_{b_j}$ is holding
         if ($p_j - w_i$) $\ge 0$ and ($q_j - h_i$) $\ge 0$ then
            $mx1 = \max(width, x_j + h_i)$
            $my1 = \max(height, y_i + h_i)$
         else if (($p_j - h_i$) $\ge 0$ and ($q_j - w_i$) $\ge 0$ then
            $mx2 = \max(width, x_j + h_i)$
            $my2 = \max(height, y_i + h_i)$
         else set $m$ to $\infty$ and goto the next step
         $m1 = \max(mx1, R * my1)$
         $m2 = \max(mx2, R * my2)$
         $m_j = \min(m1, m2)$
         if ($m1 \le m2$) then $Orient_j = 0$
         else $Orient_j = 1$
   end
If $P_{b_j}$ has two corners then choose the one which gives smaller $m_j$.
If $P_{b_j}$ does not have any corners then set $m_j$ to $\infty$.
2. Call parallel procedure FindMin ($m_{l\,l=0}{}^{m-1}, m_k b_i$)
end FindBestCorner ;
```

```
Procedure UpdateCorners ($(w_i, h_i), k$) ;
1. Processor $P_{b_k}$ remove [$(x_k, y_k), (p_k, q_k)$].
2. Each Processor $P_{b_j}$ does the following
   begin
   Let [$(x_j, y_j), (p_j, q_j)$] the corner $P_{b_j}$ is holding
   if $Orient_i = 1$ then
      $w'_i = w_i, h' = w_i$
   else
      $w'_i = w_i, h'_i = h_i$
   if ($y_k \le y_j < y_k + h'_i$) and
      ($x_j \le x_k < x_j + p_j$) then $p_j = x_k - x_j$
   if ($x_k \le x_j < x_k + w'_i$) and
      ($y_j \le y_k < y_j + q_j$) then $q_j = y_k - y_j$
   if $p_j = 0$ or $q_j = 0$ then
      remove [$(x_j, y_j), (p_j, q_j)$]
   end
If $P_{b_j}$ has two corners then update the second one in the same way.
end UpdateCorners ;
```

```
Procedure FindNewCornerSize ($(w_i, h_i), k$) ;
1. Each Processor $P_{b_j}$($j < i$) does the following
   begin
      if $Orient_i = 1$ then
         $w'_i = h_i, h'_i = w_i$
      else
         $w'_i = w_i, h'_i = h_i$
      if ($YLoc_j \le y_k \le YLoc_j + h_j$) and
         ($x_k + w'_i \le XLoc_j$) then $p_j = XLoc_j$
      else $p_j = \infty$
      if ($XLoc_j \le x_k + w'_i \le XLoc_j + w_j$) and
        ($y_k \le YLoc_j$) then $q_j = YLoc_j$
      else $q_j = \infty$
2. Call parallel procedure FindMin ($p_{l\,l=0}{}^{i-1}, p, b_i$)
3. Call parallel procedure FindMin ($q_{l\,l=0}{}^{i-1}, q, b_i$)
4. Find $p'$ and $q'$ in the same way for the other corner.
5. Store the two new corners, [$(XLoc_i + w'_i, YLoc_i), (p, q)$] and
   [$(XLoc_i, YLoc_i + h'_i), (p', q')$], at Processor $P_{b_i}$.
end FindNewCornerSize ;
```

```
Procedure Broadcast ($j, V$) ;
// Processor $P_j$ broadcasts value $V$ to all the processor in $PS$. //
```

```
Procedure FindMin ($a_{l\,l=0}{}^{m-1}, b, j$) ;
// Given the $a_l$ values with one value per processor, find the
 minimum ($b$) of these values and store it in the processor $P_j$. //
```

```
Procedure SendPacketToSource
1. Each processor $P_{b_i}$ in clude $XLoc_i$, $YLoc_i$, $Orient_i$, width and
   height in the packet $\langle (w_i, h_i), S_i \rangle$.
2. Send each packet $\langle (w_i, h_i), S_i, (XLoc_i YLoc_i), Orient_i, (width, height) \rangle$ to processor $S_i$. For this, one-to-one routing can be used [5].
```

```
end SendPacketToSource
```

After executing the above algorithm, processor S_i can find a submesh for grid (w_i, h_i) using the information included in the packet. The time complexity of the whole algorithm is analyzed as follows. Step (a) takes $O(\sqrt{N})$ time, where N is the number of processors, and step (b) takes $O(m\sqrt{m})$ time. Both procedure **FindBestCorner** and **FindNewCornerSize** take $O(\sqrt{m})$ time. procedure **UpdateCorners** takes a constant time. Hence step (c) takes $O(m\sqrt{m})$ time. Step (d) takes $O(\sqrt{N})$ time. The total time complexity of the algorithm is $O(\sqrt{N} + m\sqrt{m})$. Since we used only m processors, the actual time complexity is $O(m\sqrt{m})$.

6. Conclusions

Two-dimensional packing algorithm can be used for allocating submeshes in mesh multiprocessor systems. Previously, we developed an efficient packing algorithm called TP heuristic, and showed how the results of the packing could be used for allocating submeshes. In this paper, we presented theoretical performance bounds for TP heuristic. The bounds presented here guarantee that it will never produce solutions values that exceed a certain limit. We also presented a parallel version of the algorithm, and analyzed its time complexity. The parallel packing algorithm will consume reduced time when it is executed by multiple processors in mesh multiprocessors.

References

[1] Y. Azar, L. Epstein, "On Two Dimensional Packing," Journal of Algorithms, Vol.25, No.2, pp.290-310, November, 1997.

[2] B. S. Baker, E. G. Coffman and R. L. Rivest, "Orthogonal Packings in Two Dimensions," SIAM Journal on Computing, Vol.9, No.4, pp.846-855, August, 1980.

[3] M. Hi, R. Ouafi, "A best-first branch-and-bound algorithm for orthogonal rectangular packing problems," International Transactions in Operational Research, Vol.5, Issue 5, pp.345-356, September, 1998.

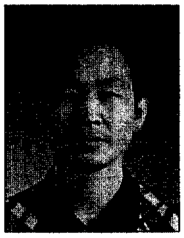
[4] I. Hwang, "Efficient Processor Allocation Algorithm Using Two-Dimensional Packing," Journal of Parallel and Distributed Computing, Vol.42, pp.75-81, 1997.

[5] F. T. Leighton, Introduction to Parallel Algorithms and Architectures, Morgan Kauf-mann Publishers, Inc. San Mateo, CA, 1991.

[6] M. Paulhus, "An Algorithm for Packing Squares," Journal of Combinatorial Theory, Series A, Vol.82, No.2, pp.147-157, May, 1998.

황 인 재

e-mail : Ih@ghost.chungbuk.ac.kr
1986년 충북대학교 컴퓨터공학과(공학사)
1981년 University of Florida. Computer & Information Sciences(공학석사)
University of Florida. Computer & Information Sciences(공학박사)
1986년~1987년 한국 전자통신연구소 연구원
1995년~현재 충북대학교 컴퓨터교육과, 컴퓨터정보통신연구소 부교수
관심분야 : 병렬처리, 병렬컴퓨터 구조, 병렬 알고리즘, 디지털 시스템설계, VLSI 설계 알고리즘



홍 동 권

e-mail : dkhong@kmucc.keimyung.ac.kr
1985년 경북대학교 전자공학과(학사)
1992년 University of Florida 전자계산학과(석사)
1995년 University of Florida 전자계산학과(박사)
1985년~1990년 한국전자통신연구원
1996년~1997년 한국전자통신연구원
1997년~현재 계명대학교 컴퓨터・전자공학부 부교수
관심분야 : 능동 실시간 데이터베이스, 병렬 처리, 성능 평가, 시뮬레이션, 멀티미디어 처리