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

Injae Hwang1·Dong-Kweon Hong1

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.

Keywords: mesh multiprocessor (Mesh Multiprocessors), Two-dimensional Packing (Two-dimensional Packing), Theoretical Performance Bound (Theoretical Performance Bound), Processor Allocation (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
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 sched-
uling 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 con-
trast, if rotations are not allowed, they showed that no al-
gorithm of constant ratio exists. For this case they designed
an algorithm with performance ratio of $O(\log \frac{1}{\varepsilon})$, where
$\varepsilon$ is the minimum width of any rectangle.

Hifi and Ouaf discovered 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 ran-
domly generated problem instances. Computational results
show that the proposed exact algorithm is able to solve small
and medium problem instances within reasonable execution
time.

Paulaus 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 alloca-
tion 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 propor-
tional 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 proces-
sors 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 $X \times Y$ pro-
cessor submesh. The grid is uniformly divided into $XY$ pi-
ces with dimension $\frac{w}{X} \times \frac{h}{Y}$, so that one can be assigned
to each processor. Then, the computational cost is propor-
tional to $\frac{w h}{X Y}$ which is the average number of grid points
assigned to each processor. Communication cost is propor-
tional 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 mul-
tiprocessors, 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
$\frac{\text{width}}{P}$, $\frac{\text{height}}{Q}$ to allocate a submesh to each grid.

The basic idea of TP-heuristic is as follows. First the gri-
ders 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 orienta-
tion 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 \( O(m^3) \).

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 \( w_i \times h_i \) 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 \( x < R_Y \) and it should be placed with long side parallel to the \( Y \)-axis if \( x > R_Y \). 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 \( x = R_Y \), 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 \( W_{\text{opt}} \) and \( H_{\text{opt}} \) be the width and height of the optimal packing and let \( W_{TP} \) and \( H_{TP} \) 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 \( W_{\text{opt}} = H_{\text{opt}} \) and \( W_{TP} \geq H_{TP} \). Then \( W_{TP} \geq \sqrt{2} W_{\text{opt}} + 3w^* \).

**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 \( W_{TP} \) and \( H_{TP} \) 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 \( W_{TP}^i - H_{TP}^i \leq w^* \) and \( W_{TP} - H_{TP} > w^* \). It also must be true that the \( i \)-th item was packed at a corner in \( R_1 \), hence \( W_{TP}^i < W_{TP} \) and \( H_{TP}^i < H_{TP} \). Let \( (x, y) \) be the location of a corner in \( R_1 \) which can accommodate the \( i \)-th item. Then \( x < y \leq H_{TP}^i \). Since \( x + w_i \leq H_{TP}^i + w^* < W_{TP} \) and \( y + h_i \leq H_{TP}^i + w^* < W_{TP} \), 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 \( \frac{(W_{TP} - 2w^*)^2}{2} \) and \( \frac{(H_{TP} - 2w^*)^2}{2} \) respectively. Note that all items in the region \( A_1 \) (or \( A_2 \)) 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 $W_{TP}^2 = 1/4((H_{TP}-2w^*)^2 + (W_{TP} - 2w^*)^2) \geq \frac{(H_{TP}-2w^*)^2}{2}$ 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 $W_{TP} \leq \sqrt{2} W_{NW} + 2w^*$.

**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 $\frac{(W_{TP} - w^*)^2}{2}$ and $\frac{(H_{TP} - w^*)^2}{2}$ 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 $W_{TP}^2 = 1/4((H_{TP} - w^*)^2 + (W_{TP} - w^*)^2) \geq \frac{(H_{TP} - w^*)^2}{2}$ 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, $w_1 \times h_1, w_2 \times h_2, \ldots, w_m \times h_m$. After the following algorithm terminates, global variables, XLoc, and YLoc, contain the location of the corner where grid $i$ is packed. Orient is set to 1 if grid $i$ was rotated and 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 $S_i$ produces a packet $\langle(w_i, h_i), S_j\rangle$ for grid $j$, where $(w_i, h_i)$ 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 $PS = P_i$, $0 \leq i \leq m-1$. The processors which produced packets send them to the processors in $PS$ (one packet per processor) using procedure TokenPacking. The processors in $PS$ will do the remaining steps of our parallel packing algorithm.

(b) Sort packets according to the packing order using a parallel sorting algorithm. After sorting, assume that processor $P_k$ is holding packet $\langle(w_k, h_k), S_j\rangle$.

(c) Store the initial free corner $(0,0)$, $(\infty, \infty)$ at processor $P_{se}$. Each processor $P_i$, 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 $P_i$ 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 $P_i$ includes the above information in its packet $(w_i, h_i, S_i)$ and sends it to $S$ using procedure SendPacketToSource.

end Parallel Packing

Procedure TokenPacking:
// Here a subset of processors $P_{j_0}, P_{j_1}, \ldots, P_{j_k}$ with $i_0 < j_1 < \cdots < j_k$ has one packet each and it is desired to store the packet of $P_{j_k}$ in $P_s$ for $i \leq k \leq (i + l) \mod m$ for some $0 \leq i \leq N - 1$.

Procedure GridPacking(i):
1. Call procedure Broadcast $(b, ((w, h), S))$.
2. Call parallel procedure FindBestCorner $((w, h), k)$ to nd the corner $(x_i, y_i)$ where the grid $(w, h)$ is to be packed, and its orientation.
3. Processor $P_i$ sets XLoc and YLoc to $x_i$ and $y_i$, respectively, and set Orient to Orient. Also $P_i$ determines the width and height of the packing after $(w, h)$ is packed.
4. Call parallel procedure UpdateCorner $((w, h), k)$ to update the sizes of corners after the grid $(w, h)$ is packed at $(x_i, y_i)$.
5. Call Parallel procedure FindNewCornerSize $((w, h), k)$ to determine the sizes of new two corners.

Procedure FindBestCorner $((w, h), k)$:
1. Each Processor $P_i$, does the following begin
   Let $[(x, y), (p, q)]$ be the corner $P_i$ is holding
   if $(p \geq w_i) \geq 0$ and $(q \geq h_i) \geq 0$ then
   $w_1 = \max (\text{width}, x_i + h_i)$
   $h_1 = \max (\text{height}, y_i + h_i)$
   else if $((p, h_i) \geq 0$ and $(q, w_i) \geq 0$ then
   $w_2 = \max (\text{width}, x_i + h_i)$
   $h_2 = \max (\text{height}, y_i + h_i)$
   else set $m = \infty$ and goto the next step
   $m_1 = \min (m_1, R + m_1)$
   $m_2 = \min (m_2, R + m_2)$
   $m_1 = \min (m_1, m_2)$
   $m_2 = \min (m_2, R + m_2)$
   if $(m_1 \leq m_2)$ then Orient $= 0$
   else Orient $= 1$
end
If $P_i$ has two corners then choose the one which gives smaller $m_1$. If $P_i$ does not have any corners then set $m_1$ to $\infty$.
2. Call parallel procedure FindMin $(m_1, m_2, P_i)$;
end FindBestCorner;

Procedure UpdateCorner $((w, h), k)$:
1. Processor $P_s$, remove $[(x, y), (p, q)]$.
2. Each Processor $P_i$, does the following begin
   Let $[(x, y), (p, q)]$ be the corner $P_i$ is holding
   if Orient $= 1$ then
   $w'_i = w_i - k_i = k_i$
   if $(x_i \geq y_i + k'_i)$ then $x'_i = x_i - k_i$
   if $(x_i \leq x_i + w'_i)$ and $(y_i \geq y_i + q_i)$ then $x'_i = x_i + k_i$
   if $(x_i \leq x_i + w'_i)$ and $(y_i \leq y_i + q_i)$ then $q'_i = y_i + k_i$
   if $p_i = 0$ or $q_i = 0$ then
   remove $[(x, y), (p, q)]$
end
If $P_s$ has two corners then update the second one in the same way.
end UpdateCorner;

Procedure FindNewCornerSize $((w, h), k)$:
1. Each Processor $P_s$, $j \in i$ does the following begin
   if Orient $= 1$ then
   $w'_i = w_i - k_i$
   $h'_i = h_i$
   if $(y_i \leq y_i + k'_i)$ and $(x_i \leq x_i + w'_i)$ then $y'_i = y_i - k_i$
   if $(x_i \leq x_i + w'_i)$ and $(y_i \leq x_i + w'_i)$ then $y'_i = y_i - k_i$
   if $(x_i \leq x_i + w'_i)$ and $(y_i \leq x_i + w'_i)$ then $y'_i = y_i - k_i$
   else $p'_i = \infty$
   if $(x_i \leq x_i + w'_i)$ and $(y_i \leq x_i + w'_i)$ then $q'_i = y_i - k_i$
   if $(x_i \leq x_i + w'_i)$ and $(y_i \leq x_i + w'_i)$ then $q'_i = y_i - k_i$
   else $q'_i = \infty$
end
2. Call parallel procedure FindMin $(p'_i, q'_i, a, b_i)$;
3. Call parallel procedure FindMin $(a, b_i, c, d_i)$;
4. Find $p'$ and $q'$ in the same way for the other corner.
5. Store the two new corners, $[(XLoc_i + w', YLoc_i), (p, q)]$ and $[(XLoc_i, YLoc_i + h'), (p', q')]$, at Processor $P_s$;
end FindNewCornerSize;

Procedure Broadcast $(j, V)$:
// Processor $P_j$ broadcasts value $V$ to all the processor in $P_s$.

Procedure FindMin $(a, b, c, d)$:
// Given the $a_i$ values with one value per processor, find the minimum $(\theta)$ of these values and store it in the processor $P_j$.

Procedure SendPacketToSource:
1. Each processor $P_s$ in clade XLoc, YLoc, Orient, width and height in the packet $((w, h), S)$.
2. Send each packet $((w, h), S, (XLoc, YLoc), Orient, (width, height))$ to processor $S$. For this, one-to-one routing can be used [5].

end SendPacketToSource

After executing the above algorithm, processor $S$ can find a submesh for grid $(w, h)$ using the information included in the packet. The time complexity of the whole algorithm is analyzed as follows. Step (a) takes $O(N)$ time, where $N$ is the number of processors, and step (b) takes $O(m^3)$ time. Both procedure FindBestCorner and FindNewCornerSize take $O(V^3)$ time. Procedure UpdateCorner takes a constant time. Hence step (c) takes $O(m^3)$ time. Step (d) takes $O(N^3)$ time. The total time complexity of the algorithm is $O(N^3 + m^3 V^3)$. Since we used only $m$ processors, the actual time complexity is $O(m^3 V^3)$.
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