An Efficient Algorithm for the Physical Mapping of Clustered Task Graphs onto Multiprocessor Architectures

Nectarios Koziris Michael Romesis

Panayiotis Tsanakas George Papakonstantinou

National Technical University of Athens
Dept. of Electrical and Computer Engineering
Computer Science Division
Computing Systems Laboratory
Zografou Campus, Zografou 15773, Greece
e-mail: {nkoziris, mromes}@cslab.ece.ntua.gr

Abstract

The most important issue in sequential program parallelisation is the efficient assignment of computations into different processing elements. In the past, too many approaches were devoted in efficient program parallelization considering various models for the parallel programs and the target architectures. The most widely used parallelism description model is the task graph model with precedence constraints. Nevertheless, as far as physical mapping of tasks onto parallel architectures is concerned, little research has given practical results. It is well known that the physical mapping problem is NP-hard in the strong sense, thus allowing only for heuristic approaches. Most researchers or tool programmers use exhaustive algorithms, or the classical method of simulated annealing. This paper presents an alternative approach onto the mapping problem. Given the graph of clustered tasks, and the graph of the target distributed architecture, our heuristic finds a mapping by first placing the highly communicative tasks on adjacent nodes of the processor network. Once these «backbone» tasks are mapped, there is no backtracking, thus achieving low complexity. Therefore, the remaining tasks are placed beginning from those close to the «backbone» tasks. The paper concludes with performance and comparison results which reveal the method's efficiency.

1. Introduction

In the last years the evolution in the fields of VLSI technology and computer networking has given raise to the utilization of distributed computing systems. Distributed computers are attractive to many demanding applications since they provide the user with modularity, scalability and low cost decentralized processing power. Nevertheless distributed computing has a sound drawback which discourages application developers to use it. The existence of many parallel processors is not fully exploited because of the interprocessor communication overhead. The degradation of optimal speedup when the number of processors increases is caused by the excessive sometimes amount of messages between non-neighboring cells.

Researchers have already focused on such low-performance problems and proposed various methodologies to the optimal parallelization of the given programs. Experience has shown that an effective solution to the general task graph scheduling problem onto a given architecture is a multistep approach which includes the task graph description of the problem, efficient scheduling of the tasks onto a virtual fully connected parallel machine, merging tasks into larger clusters and finally assigning clusters into the physical processor topology.

As far as the task graph scheduling with precedence constraints is concerned, the most widely used model is the directed acyclic graph with edges and nodes having various weights. Node weights represent computation time for the corresponding tasks, while edge weights are communication requirements between tasks. Each edge direction gives a set of precedence constraints which should be preserved. The general task graph scheduling problem with communication delays is NP-COMPLETE as presented in [9]. Sarkar [10], Gerasoulis and Yang [3] presented good heuristics for solving the task graph scheduling problem with arbitrary communication costs.

All previous researchers were supposing unbounded or bounded number of processors with CLIQUE topology.

This hypothetical topology is far away from the realistic case. Modern distributed memory parallel systems are organized in different topologies, including hypercube, mesh or ring architectures. There is a strong need for efficient placement of clustered tasks into the real processor network. The mapping of virtual into real topologies is called the physical mapping problem or the task allocation problem.

The task allocation problem can be optimally solved in special cases such as two-processor distributed systems, or linear array of any number of processors. If the target architecture contains two processors, then the task allocation problem is stated as a maximum flow minimum cut problem [11] which can be polynomially solved using, for example, the Ford-Fulkerson algorithm. There also exists a heuristic presented [7], which addresses the general m-processor problem using the 2-way min cut algorithm m times.

Most of the theoretical work on mappings considers structured graphs like grids, hypercubes, trees, etc [8]. An increasing number of applications demand methods dealing with irregular graphs. The general mapping problem is unfortunately NP-complete, thus allowing only for efficient heuristics.

Our paper deals with the intractable problem of cluster allocation onto parallel architectures, by proposing an alternative approach to the above mentioned methodologies. We model any target parallel architecture by a non-directed graph with unitary edge weights only. This representation retains the meaningful only details of the target. Next, the graph of tasks is depicted as a non-directed graph with various edge costs and node weights resulting from clustering heuristics as the one proposed by Gerasoulis in [3]. The task assignment procedure starts by tracing the clusters which are most likely to be placed in neighboring places, where the maximum number of links is available. This step ends by placing these backbone clusters into these places. For each of the backbone clusters, the algorithm creates sets of neighboring clusters which are candidate for placement into the adjacent cells of every preassigned backbone cluster. The algorithm ends when all clusters are assigned.

In the remainder of this paper the following are presented and further elaborated: Section II reviews the multistep approach of task graph scheduling with precedence constraints using a specific topology distributed architecture. Section III presents the proposed graph model for the parallel architecture and the utilized cost function for evaluating different physical mappings. In section IV we outline the steps of the proposed algorithm. Finally section V the proposed algorithm is compared to other approaches in terms of efficiency and the outperforming results are shown.

2. The Multistep Approach

The general scheduling problem of an arbitrary task graph with communication delays onto a fixed size and connection pattern distributed architecture is NP-COMPLETE. El-Rewini et H. Ali in [1], [2] proved this NP completeness by representing the problem of task allocation onto distributed system with a split graph. The task allocation is therefore equivalent to a weighted clique graph partitioning which is NP-COMPLETE, thus proving inherent intractability. In order to find efficient methods, researchers have followed a multistep approach, where each step addresses a limited instance of the general problem. The successive steps are outlined as follows: *task clustering*, *cluster merging* and *physical mapping.*

2.1. Task Clustering - Scheduling a task graph with communication delays onto a bounded/unbounded CLIQUE of processors.

First the task graph with computation/communication costs and precedence constraints is scheduled onto a fully connected network of processors. The classical CLIQUE architecture is therefore used as a target having limited or unlimited number of processors [4]. Researchers, in this first step, propose algorithms which minimize the maximum makespan, disregarding the actual processor's topology. Even when the makespan metric is to be minimized, the scheduling problem remains NP-COMPLETE in the majority of general cases. Papadimitriou et Yannakakis in [9] have proved that the classical scheduling problem of a task graph with arbitrary communication and computation times is NP-COMPLETE. They proposed a 2-optimal approximation algorithm. In addition to this, Sarkar in [10], Gerasoulis in [3] proposed faster heuristics with acceptable performance. All these algorithms perform the same initial step: Cluster the tasks into large nodes, so that the grain of the parallelism is increased and the use of distributed processors is minimizing the task graph makespan. The resulting topology, by applying this initial step is a graph of clusters. Inside every cluster there exist several tasks which are time scheduled in the same processor and have zero intracommunication overhead. As far as clusters intercommunication is concerned, this is the summation over all tasks intercommunication overhead for all clusters. This organises the set of tasks into clusters with the following property: Each cluster contains all tasks which are to be executed on the same processor. This step is called *task clustering*.

2.2. Cluster Merging into p physical clusters

In this step, the set of clustered tasks is mapped onto a clique of bounded number of processors. Since the set of clustered tasks is larger than the number of available processors, this step assigns two or more clusters to the same processor. Sarkar in [10] has proposed a scheduling heuristic with O(|V|(|V|+|E|)) complexity, where |V| stands for the number of nodes and |E| for the number of edges. A lower complexity heuristic which is used in PYRROS [12] is the work profiling method. It merges clusters which have approximately the same arithmetic load.

2.3. Physical Mapping of p physical clusters onto p network connected processors.

While so much effort has been done concerning the first steps, the final step of cluster allocation onto the physical processor topology has not given considerable attention. Few researchers such as Bokhari have presented some heuristics. Many scheduling tools which such as OREGAMI or PYRROS or PARALLAX use heuristics or approximation algorithms which are sometimes efficient. For example Gerasoulis et Yang in Pyrros [12] use Bokhari's heuristic which is based onto simulated annealing. This algorithm starts from an initial assignment, then performs a series of pairwise interchanges by reducing the cost function and stops after $O(p^3)$ steps.

In Oregami's [6] MAPPER, a tool for task allocation onto distributed architectures, a greedy heuristic is used, called the NN-Embed. It currently supports only mesh and hypercube processor networks thus limiting its potential use. Even in the case of those two architectures, it uses a rather simple method, by listing all edges in ascending order of their weights and assigning them to the processor's network edges.

2.4. The NN-Embed Algorithm

Given a task graph, it first constructs a list of all the edges in the graph, sorted by weight. The heuristic then traverses this list in linear time and for each edge, assign endpoints as follows:

- If both nodes have already been assigned, do nothing
- If only one node has been assigned, then assign the other node to the closest free processor
- If neither node has been assigned, randomly choose a free processor and assign one node to it and the other to its closest free neighbor.

The sorting step needs O(|E|log|E|) time, and the rest is O(|E|). It is a fast heuristic, but it is limited to hypercube and mesh topologies.

Obviously, it has poor performance in many cases because it is based on single-edge adjacency, and it does not take into account any set of adjacent nodes. In Example 5, we apply both our method and the NN-Embed, and the outperforming results of the proposed heuristic are shown.

In PARALLAX [5], all task allocation heuristics assume fully connected networks of processors. Only the mapping heuristic considers arbitrary processor interconnection topology. It uses a modified list scheduling technique with priorities, by taking into account the communication delays. The node's priority in the list is its level.

In [8], Monien and Sudborough reviewed results on mapping specific task graphs into the most popular parallel architectures.

3. Graph Models

Our problem considers two graphs: The graph of virtual processors, or equivalently the graph of clustered tasks, defined as G_c (V_c, E_c) and the graph of physical processors, defined as G_p (V_p, E_p).

Definition 3.1: Consider two graphs and G_w (V_w, E_w) where V is the set of nodes and E is the set of edges with $|V_w| = |V_p|$, we define the following function $F_m{:}V_w \rightarrow V_p$ as the physical mapping function :

$$\forall\ v_w, v_w' \in V_w \text{ with } (v_w, v_w') \in E_w,$$
$$\exists\ (F_m(v_w), F_m(v_w')) \in E_p$$

Let us define some topological parameters for the processor graph/network:

- *Hop* is the unit distance between any two directly connected processors in the network
- *Distance* between two processor nodes u and v is called the number of hops in the shortest path connecting u and v. *Distance* is represented by dist(u,v).

The following formula defines the cost function used to evaluate the different mappings:

Cost Function for a mapping F_m:

$$CF(F_m) = \sum_{(v_w, u_w) \in E_w} dist(F_m(v_w), F_m(u_w))\, x\, comm(v_w, u_w)$$

where:

- dist($F_m(v_w)$, $F_m(u_w)$) is the shortest path in the processor graph between the 2 processors-nodes where the tasks v_w and u_w are mapped.
- comm(v_w, u_w) is the total communication cost between the tasks v_w,u_w

The following properties characterize the efficiency of a mapping from G_w to G_p:

Edge-dilation is the maximum distance in G_p an edge of G_w has to be routed. Formally:

Edge-congestion is the maximum number of edges from G_w routed via an arbitrary edge (or node) of G_p

In most theoretical works dealing with the physical mapping problem, minimizing edge dilation and edge congestion are the primary goals. Our objective is therefore to find the mapping F_m which minimizes CF_m, or formally:

Definition 3.2: A mapping F_m^{opt} with respect to Cost Function CF() is called optimal if:

$CF(F_m^{opt}) = \min \{ CF(F_m) \mid F_m \in MAP\}$,

where MAP is the set of all possible mappings

3.1. Example

Consider the G_c (V_c,E_c) with c_{ij} costs shown in figure 1, and the processor graph $G_p(V_p,E_p)$:

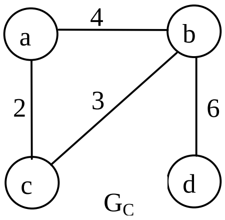

Figure 1. The Cluster Graph

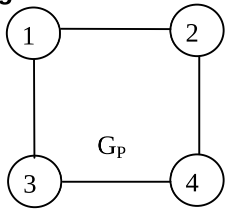

Figure 2 : The target architecture

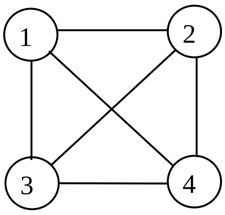

Figure 3: CLIQUE architecture

Obviously there exist 4! alternative mappings which can be found by exhaustive search. For example, consider the following alternative mappings and evaluate the corresponding cost function $CF_m()$:

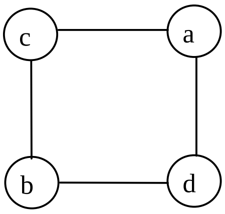

Figure 4: Mapping F_{M1}

$CF_{M1}= 2 + 4x2 + 6 + 3=19$

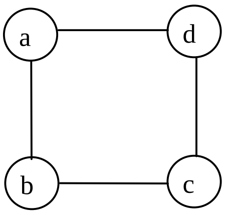

Figure 5: Mapping F_{M2}

$CF_{M2}= 4 + 2x2 + 3 + 6x2 = 21$

Obviously, F_{m1} is better than F_{m2} in terms of total completion time, since it imposes less communication overhead than F_{m2}. The least communication is achieved by mapping F_{m3} shown below:

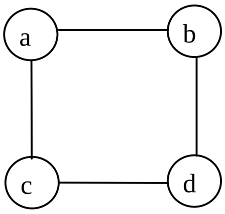

Figure 6: Optimal Mapping F_{M3}

$CF_{m3} = 2 + 4+ 6 +2x3 = 18$

4. The Physical Mapping Algorithm (PMAP)

This heuristic tries to find the most communication intensive task-nodes and map them and their neighbors into neighboring processing nodes on the processor graph. At the beginning, the processor graph is analyzed and, for each node, the number of adjacent nodes is calculated. Subsequently, the nodes of the task graph are sorted by ascending order of their total communication weight and number of neighbors. The heuristic then places the most demanding task-nodes in terms of total communication links and cost to the respective nodes of the processor graph. Once these core task nodes are placed, there is no backtracking. Next, the heuristic places the adjacent of the core nodes to adjacent cells of the processor graph, by making locally best-fit comparisons.

The algorithm performs the following steps:

4.1. First Phase:

1st step: Adjust the maximum number of neighbors in the task-graph to the maximum number in the processor graph, by removing the less communication cost edges for each task-node.

2nd step: Sort the nodes of the processor graph by ascending order of their neighboring links and the nodes in the cluster graph G_c by the number of communication links they need.

3rd step: Assign the most demanding task-node to the best suitable node of the processor graph and his neighboring nodes to the best suitable neighboring nodes of the processor graph.

4.2. Second Phase:

4th step: Place back the edges removed at the first step.

5th step: From the processors not allocated yet find the one who neighbors with the most processors already assigned.

6th step: Find a task that is distanced by at most i (at the beginning i=1) from all the tasks that are mapped to the neighboring processors of the processor found on the previous step. If found assign this task to the processor of the 5th step.

7th step: Repeat steps 5-6 until no more assignments can be made. If there are more processors to be allocated, increase i by one and go back to the 5th step.

4.3. Details of the algorithm

```
1st Phase:
Adjust_neighbors($G_c$);
$Adj(v) = \{u: (u,v) \in E_c\}$
/* Placement of the most communication costful node */
find $v_c \in G_c$ : $rank(v_c) = \max rank(v)$
find $v_p \in G_p$ : $rank(v_p) = \max rank(v)$
$F_m(v_c) = v_p$;
/* Placement of neigbors to neigboring cells */
  while $Adj(v_p) \neq \varnothing$
      begin
            find $v_{ac}$ : $rank(v_{ac}) = \max rank(v_a)$
                        $Adj(v_c) = Adj(v_c)-\{v_{ac}\}$
            find $v_{ap}$ : $rank(v_{ap}) = \max rank(v_p)$
                        $F_m(v_{ac}) = v_{ap}$
            PROCS_ALLOC = PROCS_ALLOC ∪ {$v_{ap}$}
            PROCS_LEFT = PROCS_LEFT-{$v_{ap}$}
          TASKS_LEFT = TASKS_LEFT-{$v_{ac}$}
      end
2nd phase:
/* Placement of all other nodes */
Place_back_edges;
 i=1
while PROCS_LEFT ≠ ∅
            begin
                  assigning=false
                  for all u ∈ PROCS_LEFT
                     begin
                        find u ∈ |Adj(u)∩ PROCS_ALLOC| = max
                       NEIGH = { $v_i$: $F_m(v_i) \in$ (Adj(u) ∩
                                 PROCS_ALLOC)}
                          /*set of tasks which have already been
                          assigned to neighboring procs*/
                      $Adj(1,v_i)=Adj(v_i)$
                      $Adj(i,v_i)=Adj(i-1, v_i) \cup$
                                 $\{u:(u,w) \in E_C, w \in Adj(i-1,v_i)\}$
                     let CAND=∩$Adj(i,v_i)$∩TASKS_LEFT
                  if CAND ≠ ∅
                           begin
                              find $v_{cand}$ : $rank(v_{cand}) = \max rank(v)$,
                              $v \in$ CAND
                              $F_m(u)=v_{cand}$
                             PROCS_LEFT=PROCS_LEFT-{u};
                            PROCS_ALLOC =
                                   PROCS_ALLOC∪{u};
                            TASKS_LEFT = TASK_LEFT-{$v_{cand}$}
                            assigning=true
                           end
            end /* of for*/
            if assigning=false then i=i+1
            /* i increases when during the previous cycle there has been
no assignment */
end
Adjust_neighbors routine:
neigh_card=max |Adj(v)|, $v \in G_p$
For each $v \in G_c$ : |Adj(v)| > neigh_card
            repeat
                find u ∈ Adj(v): comm(v, u) = min comm(v, w) w∈
Adj(v)
                  Adj(v)=Adj(v)-u
            until        |Adj(v)| = neigh_card
end of for;
```

5. Example

Consider the G_c (V_c, E_c) with c_{ij} costs shown in figure 7, and the processor graph $G_p(V_p,E_p)$ in figure 8:

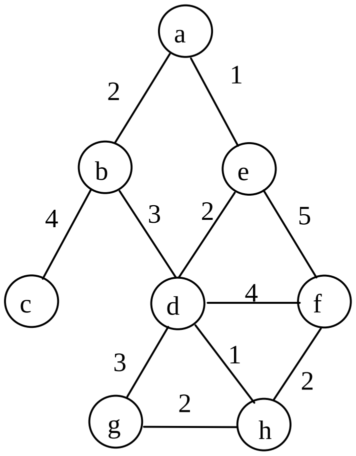

Figure 7: The Cluster Graph for Example

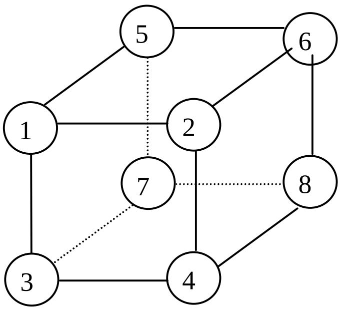

Figure 8: The target architecture for Example

5.1.First Phase

Since # of Neighbors of d > max Neigh we adjust the task graph by deleting edges : (d,h) = 1, (d, e) = 2.

Task	# of Neighbors	Total comm.	ranking
a	2	3	7
b	3	9	3
c	1	4	8
d	3	10	2
e	2	6	4
f	3	11	1
g	2	5	6
h	2	4	5

Table 1: Statsistics for the task graph

V_p=1,V_c=f
F_m(f)=1
Adj(f)={d, e, h }
F_m(d)=2
F_m(e)=3
F_m(h)=5

TASKS_LEFT={a,b,c,g}

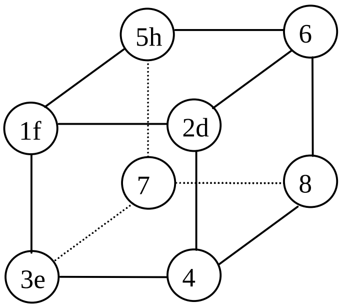

Figure 9: Assignments after the 1st phase

5.2 Second Phase:

insert back edges : (d,h)=1,(d,e)=2
1st cycle i=1
u_1=4, |Adj(4) ∩ *PROCS_ALLOC*|= 2
u_2=6, |Adj(6) ∩ *PROCS_ALLOC*|= 2
u_3=7, |Adj(7) ∩ *PROCS_ALLOC*|= 2
u_4=8, |Adj(8) ∩ *PROCS_ALLOC*|= 0
Let u = 4:
NEIGH = {d,e}
CAND=Adj(d)∩Adj(e)∩TASKS_LEFT=∅

Let u = 6:
NEIGH={d,h}
CAND=Adj(d)∩Adj(h)∩TASKS_LEFT={g}
F_m(g)=6
2nd cycle i=1
u_1=4, |Adj(4) ∩ *PROCS_ALLOC*|=2
u_2=7, |Adj(7) ∩ *PROCS_ALLOC*|=2
u_3=8, |Adj(8) ∩ *PROCS_ALLOC*|=1
Let u = 4:
NEIGH = {d,e}
CAND =Adj(d)∩Adj(e)∩TASKS_LEFT=∅
Let u = 7:
NEIGH = {h,e}
CAND=Adj(h)∩Adj(e)∩TASKS_LEFT=∅
Let u = 8:
NEIGH={g}
CAND=Adj(g)∩TASKS_LEFT=∅
No assignment during this cycle: i increases by one

3rd cycle i=2
u_1=4, |Adj(4) ∩ *PROCS_ALLOC*|=2
u_2=7, |Adj(7) ∩ *PROCS_ALLOC*|=2
u_3=8, |Adj(8) ∩ *PROCS_ALLOC*|=1
Let u = 4:
NEIGH = {d,e}
CAND = Adj(2,d) ∩ Adj(2,e) ∩ TASKS_LEFT = {b,a}
F_m(b)=4

4th cycle i=2
u_1=7, |Adj(7) ∩ *PROCS_ALLOC*|=2
u_2=8, |Adj(8) ∩ *PROCS_ALLOC*|=2
Let u = 7:
NEIGH = {h,e}
CAND=Adj(2,h)∩Adj(2,e)∩TASKS_LEFT = ∅
Let u = 8:
NEIGH= {b,g}
CAND=Adj(2,b)∩Adj(2,g)∩TASKS_LEFT = ∅
No assignment during this cycle: i increases by one

5th cycle i=3
u_1=7, |Adj(7)∩*PROCS_ALLOC*|=2
u_2=8, |Adj(8)∩*PROCS_ALLOC*|=2

Let u = 7:
NEIGH={h, e}
CAND=Adj(3,h)∩Adj(3,e)∩TASKS_LEFT={a,c}
$F_m(a)=7$
$F_m(c)=8$

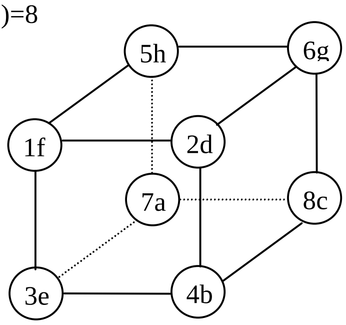

Figure 10: Assignments after the 2nd phase – PMAP result

5.3. NN-Embed algorithm

Sorting of Edges in descending order of their weight:

(e,f)	5	(a,b)	2
(c,b)	4	(d,e)	2
(d,f)	4	(g,h)	2
(b,d)	3	(h,f)	2
(g,d)	3		

Table 2: Weight of edges

Assign (e,f) edge to an arbitrary edge, e.g: $F_m(e)=1$, $F_m(f)=5$
Assign (c,b) edge to an arbitrary edge, e.g: $F_m(c)=2$, $F_m(b)=6$
Assign (d,f) edge: Since f is already assigned, assign d to the closest free neighbor: $F_m(d)=7$
All nodes of (b,d) are assigned.
Assign (g,d) edge: Since d is already assigned, assign g to the closest free neighbor: $F_m(g)=3$
Assign (b,a) edge: Since b is already assigned, assign a to the closest free neighbor: $F_m(a)=8$
All nodes of (e,d) are assigned.
Assign (g,h) edge: Since g is already assigned, assign h to the closest free neighbor: $F_m(h)=4$
All nodes of (h,f) are assigned.

The following table compares the performance of the algorithms for this example:

Algorithm	**Cost**
NN_embed	39
PMAP	34
Optimal	33

6. Performance Results

We report our experiments on the PMAP algorithm. For the sake of comparisons we have created a program which, given the task and the processor graphs, generates all possible mappings and calculates the total communication time based on the metric of section III. We have also created a random task graph generator with various communication-edge weights and number of tasks-nodes and a graph generator based on fixed topologies. We have also applied the PMAP heuristic and the NN_Embed heuristic on the same graph pairs. The results obtained are shown in the table below. We can see that the PMAP algorithm gives better results than the NN_Embed algorithm, especially when the task and processor graphs have fixed topologies. But even when the graphs have been randomly created PMAP outperforms NN-Embed by more than 10%.

Nodes	Task topology	Processor topology	PMAP vs. NN_Embed	optimal vs. NN_Embed
5-10	Random	Random	13%	26%
10-20	Random	Random	12%	-
20-40	Random	Random	11%	-
32	Ring	Mesh(8x4)	29%	-
32	Mesh(8x4)	Ring	28%	-
32	Mesh(8x4)	Mesh(8x4)	39%	-
32	Mesh(8x4)	Hypercube	10%	-
32	Hypercube	Mesh(8x4)	21%	-
10	Ring	Mesh(5x2)	32%	42%
10	Ring	Binary Tree	10%	20%
10	Mesh(5x2)	Ring	14%	24%
10	Mesh(5x2)	Mesh(5x2)	31%	42%
10	Binary Tree	Ring	20%	30%
10	Binary Tree	Mesh(5x2)	21%	33%

Table 3: Performance results

7. Related Work

There are not many systems which tackle with the problem of task allocation on specific target architectures, thus solving the physical mapping problem. PYRROS, developed by Yang and Gerasoulis, uses the heuristic of Bokhari which is based on simulation annealing. In OREGAMI [6] task allocation is performed in three steps: contraction of the task graph to a smaller graph, assignment of the contracted clusters of tasks to processors and finally routing of messages through the interconnection network to minimize contention. In OREGAMI, there exist canned algorithms for typical interconnection patterns concerning the target parallel architectures, which have been developed *a priori*. If the interconnection pattern is not in the above library, the NN-Embed greedy heuristic is used.

PARALLAX [5], is another scheduling tool which contains a task allocation heuristic assuming a specific interconnection topology for the target architecture.

8. Conclusion

In this paper a new approach was presented for the physical mapping of task graphs into parallel architectures having arbitrary interconnection topologies. It is obvious the general problem is intractable thus allowing only for efficient heuristics. The proposed algorithm has a low complexity and gives very good results for graphs with equilibrated communication loads between adjacent tasks. This new algorithm was implemented and tested, and the outperforming results were reported. Future work includes the introduction of additional criteria which would increase the heuristic's performance as far as non-counterbalanced graphs are concerned.

9. Acknowledgements

This research was supported in part by the Greek Secretariat of Research and Technology (GSRT) under a PENED 99/308 Project.

References

[1] H. Ali et H. El-Rewini. Task Allocation in Distributed Systems: A Split Graph Model. *Journal of Combinatorial Mathematics and Combinatorial Computing*, vol. 14, pp. 15-32, October 1993.

[2] H. El-Rewini, T. G. Lewis and H. Ali. *Task Scheduling in Parallel and Distributed Systems.* Prentice Hall, 1994.

[3] A. Gerasoulis and T. Yang. On the Granularity and Clustering of Directed Acyclic Task Graphs. *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 6, pp. 686-701, Jan. 1993.

[4] N. Koziris, G. Papakonstantinou and P. Tsanakas. Optimal Time and Efficient Space Free Scheduling for Nested Loops. *The Computer Journal, vol. 39, no 5, pp 439-448, 1996*.

[5] T. Lewis and Hesham El-Rewini. Parallax: A Tool for Parallel Program Scheduling. *IEEE Parallel & Distributed Technology*, vol. 1, no. 2, May 1993, pp 62-72.

[6] V. Lo, S. Rajopadhye, S. Gupta, D. Keldsen, M. Mohamed, B. Nitzberg, J. Telle and X. Zhong. OREGAMI: Tools for Mapping Parallel Computations to Parallel Architectures. *Int'l Journal of Parallel Programming*, vol. 20, no. 3, 1991, pp. 237-270.

[7] V. Lo. Heuristic Algorithms for Task Assignment in Distributed Systems. *IEEE Trans. Comput.*, vol. C-37, no. 11, pp. 1384-1397, Nov. 1988.

[8] B. Monien And H. Sudborough. Embedding one Interconnection Network in Another. *Computational Graph Theory, Springer-Verlag, Computing Supplement 7*, pp 257-282, 1990.

[9] C. H. Papadimitriou and M. Yannakakis. Toward an Architecture-Independent Analysis of Parallel Algorithms**.** *SIAM J. Comput.*, vol. 19, pp. 322-328, 1990.

[10] V. Sarkar. *Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors.* Cambridge, MA: MIT Press, 1989.

[11] H. Stone. Multiprocessor Scheduling with the Aid of Network Flow Algorithms. *IEEE Trans. Soft. Engin.*, vol. SE-3, no. 1, pp. 85-93, Jul. 1977.

[12] T. Yang and A. Gerasoulis. PYRROS: Static Task Scheduling and Code Generation for Message Passing Multiprocessors. *Proc 6th Int'l Conf. Supercomputing (ICS92),* ACM Press, New York, N. Y., 1992, pp. 428-437.