Mapping Resource Constrained Applications on Chip Multiprocessors

Anilkumar Nambiar
Dept of Computer Science,
Wayne State University
Detroit, MI, 48202

Vipin Chaudhary
Institute for Scientific Computing,
Wayne State University
Detroit, MI, 48202

Abstract - Most of the semiconductor companies are targeting multi-core processor for high-end embedded systems. Programming such processors for performance is quite cumbersome without proper development tools. Conventional mapping algorithms fail to address the issues of tight resource constraint found in embedded applications running on these processors. We present resource constraint based application-mapping heuristics for high performance multi-core embedded architectures. Experimental results using synthetic tests and real programs such as MPEG-2 Audio Video Decoder (MAVD) confirm the superiority of the proposed heuristics over existing solutions. For validation we have considered Cradle Technologies' MDSP as the target multi-core processor.

Keywords: Mapping, Scheduling, Chip-Multiprocessors, System-On-Chip, High-Performance Embedded systems, System level Design.

1.0 Introduction

An interesting trend in today's high-end embedded systems is the use of system-on-chip architectures. Most of these systems use homogenous multiprocessors, while some use heterogeneous multiprocessors. The heterogeneous multiprocessors use a combination of RISC and DSP, as most of the applications today require both to varying degrees. MDSP chip developed by Cradle Technologies is one such example of clustered heterogeneous multiprocessor. These multi-core chips provide very high raw computational power. To harness this power one requires specialized tools to map and schedule applications. Without appropriate tools, the system development task is cumbersome at best and impossible at worst. We'll target the MDSP architecture for this research, as most of the multi-core heterogeneous systems have similar architectures. In MDSP, each cluster is made up of RISC and DSP processors. The main goal of such clustering is to reduce the bandwidth requirement of the processors. Each cluster has its own set of resources such as memory, semaphore, timers, etc. The access to a resource within the cluster takes less clock cycles as compared to any access outside the cluster, as the request has to be served through a shared global bus. The desirable implementation on this system would be to reduce access outside the cluster in order to improve performance. When you have a small system with one or two cluster, with each cluster having 12 processors as in the case of MDSP, the task of mapping applications is manageable. One can identify the resource requirements manually and map the application appropriately. But as soon as the number of clusters increases, the complexity of mapping applications increases. Rather than struggling to find an optimum-mapping scheme, the programmer could spend more time on development of the applications and leave the details of mapping to the tools.

A mapping problem can be transformed into a scheduling problem and there has been a lot of research in scheduling theory[1]. We have developed a new heuristic to map tasks on the MDSP chip and have found better results over the existing heuristics.

Rest of the paper is organized as follows. Section 2 gives the problem definition; section 3 presents related work in scheduling theory; section 4 discusses the heuristics developed; section 5 presents experimental results; and section 6 has the concluding remarks.

2.0 Problem Definition

The MDSP is a heterogeneous multi-core chip developed by Cradle Technologies [2]. The Cradle CRA2003 chip has a hierarchical architecture, with clusters within a chip. Each cluster or Quad consists of 4 RISC processors or PEs, 8 DSP co-processors or DSEs, 64 Kbytes of data memory, 4 channel DMA engines and other resources such as semaphores, and timers. Each

Program and its Resources	**Audio Decoder**	**Audio Renderer**	**Video Decoder**	**Video Renderer**	**TS Parser**	**System Controller**	**TS Feeder Simulator**	**ISR Simulator**	**Timer Simulator**
Computation	1 PE 1 DSE (1 data DSE)	1 PE 1 DSE	6 PE 12 DSE	1 PE 2 DSE	1 PE 1 DSE	1 PE	1 PE	1 DSE	1 DSE
Private Local Memory	0x22D0	0xDD0	0xD68	0xBF8	0xED8	0x2F0	0x78	-	-
Shared Local Memory	-	-	0x9C0	-	0x8	-	-	-	-

Fig. 1 MAVD System Resource Requirement Table

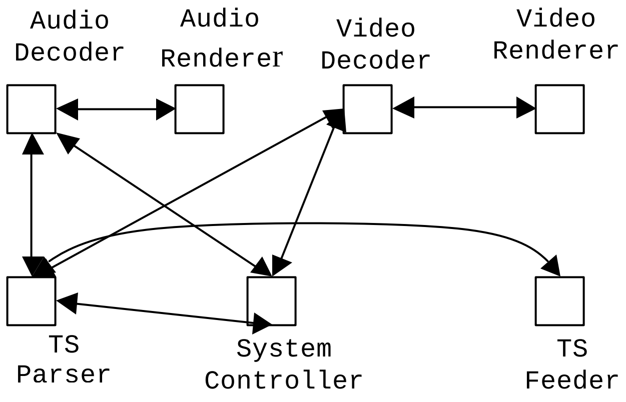


Fig. 2 MAVD System Communication Graph

chip may have varying number of clusters from 1 to 5. If one wants to design an MPEG-2[3] Audio Video Decoder (MAVD) system using the MDSP chip, one needs to decide on the resource requirements for each task. A typical MAVD system will consists of following tasks: Audio Decoder, Video Decoder, Audio Renderer, Video Renderer, TS Parser, System Controller, TS Feeder Simulator. The memory and processor requirement is given in the Fig. 1[1]

The above tasks are developed independently and need to be integrated. Some of the tasks are independent, while others have inter-task communication. The first requirement would be to allocate all the processors for a given task in the same cluster, as most of the tasks will have more intra-task communication than inter-task communication. In addition to intra-task communication, the tasks will communicate with the allocated resources to varying degree. The resources in this case are memory, and co-processor. If there were more communication between co-processor and processor than inter-task communication, it would be better to allocate the co-processor from the same cluster in which the task is running, and then try to see if the two tasks involved in communication can be allocated to the same cluster. So one assigns weight to the amount of communication for each resource type. This forms the basis for our heuristics. The communication graph for the video decoder is depicted in Fig. 2.

The solution to the MAVD system is presented at the end of this paper.

The problem could be formally stated as follows:

Given a

a) List of tasks T = $<t_1, t_2, \ldots.. t_n>$
b) Task priority P = $<p_1, p_2, \ldots\ldots p_n>$
c) Fixed Resource requirement R_{ik}, where i is the task id and k is the resource id
d) $\sum_{i=1}^{n}\sum_{k=1}^{m} R_{ik} \le R'$ where R' is the total resource available in the System,
e) and Communication $C_{ij}= \{1,0\}$, where i and j are task ids, is 1 if there is inter-task communication, 0 otherwise

Find a mapping to reduce inter-cluster communication.

3.0 Related Work

Considerable research has been done in the scheduling theory [4]. We would like to show why the solutions from various researches couldn't be directly applied to solve this problem. There have been solutions to problem with malleable tasks [5] [18] [19]. Shachnai and Turek [6] consider architecture similar to the one we are using. But the applications that we consider are not malleable and have strict

[1] The figures were obtained from the MPEG-2 AV decoder developed at Cradle Technologies.

resource requirement. Shrivastav [14] gives a polynomial time algorithm for multi-resource constraint scheduling. But it considers the resource to be of unit size and renders it unusable to us in its original form. Authors from the following paper [7] [15] [16] and [17] propose few generalized mapping algorithm to schedule directed acylic graph and lack solution for resource constraint. Solutions from other areas of research such as bin-packing, graph coloring, and linear programming are used for solving this problem. The mapping problem under consideration is similar to bin-packing problem. The bin-packing problem can be stated as follows:

A list $L = <a_1, a_2 \ldots a_n>$ of items must be packed into, i.e. partitioned among a minimum cardinality set of bins B_1 B_2 subject to the constraint that the set of items in any bin fits within that bin's capacity.

Here the clusters are bins and the applications need to be packed into these bins. The above statement requires the task not to be split across bins, but we relax the constraints by splitting the tasks across bins at the cost of communication overhead. Bin-packing has been proved to be NP-hard [8]. There has been sufficient research to obtain good approximate solutions to this problem [9]. These heuristics fail to consider the resource constraint. In the basic version of this problem the constraint could just have been the number of processors in a cluster and the processor requirement of the application. Then one could have just used one of the existing solutions for offline bin packing and applied them to obtain a solution. Our problem has multiple constraints, i.e. processors, memory, semaphores, timers, communications, etc. Most of the multicapacity bin-packing algorithms [10] are extensions to the single capacity bin-packing algorithm. These heuristics in their original form do not satisfy our requirements. The same applies for d-capacity Best Fit, Next Fit or Any Fit heuristic. We relax the requirement of placing all the sub-items in the same bin. In our case processors can be in one cluster and resources can be in other at the cost of communication overhead. The second reason is that the above classical algorithms do not deal with the affinity between items or tasks. In our case the inter-task communication requires some tasks to be in the same bin to improve performance. There are multi-capacity bin-packing algorithms [11], which improves over the classical algorithms. Inter-communication between the tasks and allocated resources requires the tasks and the resources to reside in the same bin. This renders these algorithms useless for the kind of task-graph we were mapping. We relax our assumptions a bit by not considering hard real time system scheduling. There have been polynomial time approximation heuristics for resource constrained scheduling [12] [13]. They use first fit decreasing technique (FFD) and have shown it to be (s + 1/3)-factor approximate. Since it's a simple and efficient heuristic we used it as the base line for our heuristic and have improved on it.

4.0 Improved First Fit Decreasing Algorithm (IFFD)

Our proposed heuristic improves on the existing First Fit Decreasing (FFD) algorithm [9]. Rather than starting at a random initial state, we use FFD to map the tasks and then use that as the initial state and input to our heuristic. The initial state of the mapping application on processor is important. If one changes the initial state the resulting output is completely different. Here the key for sorting is the processor requirement of the application. The other resources are excluded at this stage. Since we use a priority queue, if there are m clusters then this will require $O(nlogn*m)$ time, where n is the number of tasks to be mapped. So, in worst case it would be $O(n^2 logn)$, where the number of tasks is equal to the number of clusters in the chip. Once the tasks are mapped using FFD, we form the initial state for the processor graph G_p. The processor graph G_p is an undirected graph and is defined as follows:

$G_p = \{(V, E) \mid <u, v> \in E$ *iff the tasks v and u are allocated to the processors in the same cluster*$\}$

In FFD, we allocate the processors to the tasks based on the priority of the tasks. We start with the highest priority task and try to fulfill its requirement by allocating the required number of processors. Assigning a higher priority to a task not only makes sure that the resource requirement of the task is fulfilled before any other task is considered but it also helps in breaking ties. The allocation of processors is done from the same cluster, as it reduces intra-task communication delay. Once the initial state mapping is completed we apply our heuristic to improve on that mapping. A processor graph G_p is constructed for the initial mapping. In addition, for each class of resource R_k used by the application, we construct a constraint graph G_k. The constraint graph G_k is undirected graph and is defined as follows:

$G_k = \{(V, E) \mid \langle u, v\rangle \in E$ *iff the two tasks v and u can reside in the same cluster*$\}$

There are two types of constraints, the communication constraint and the resource constraint. In a graph, which represents the communication constraint, there is an edge between two nodes if there is communication between those two tasks. In case of a resource constraint graph, there is an edge between two nodes if the resource requirement for those two nodes satisfies the criteria of being able to be accommodated in the same cluster. In our example we have considered co-processor and local memory as resource constraints. One can include additional constraints.

Two tasks can reside in the same cluster *iff*

$R_k \geq R_{uk} + R_{vk}$

Here, R_k = Total Resource of type k available in a given Cluster
R_{ik} = Resource requirement of task i
R_{jk} = Resource requirement of task j

But, in fulfilling any constraint we always try to fulfill the primary constraint where the processors for any given task will be confined to a single cluster. In other words, our heuristic tries to avoid splits within tasks over clusters.

The user assigns weight W_i to the constraints for each task. The weight assigned depends on its importance in that node. To make this notion clear, consider the case when there is more communication with a co-processor than inter-task communication. In this case one assigns more weight to the co-processor constraint and less weight to the inter-task communication constraint. Each node in the processor graph is assigned a value based on the number of constraints satisfied for that node. The value of each node can be determined as follows:

$$W_i = \sum_{k=1}^{m} C_k * S$$

C_k is the weight assigned for resource class R_k. So higher the value, the more important it is to satisfy that constraint. S is a Boolean value, which is either *1* or *0*, as the resource requirement for some may not be present.

Once we assign a weight to each node, we calculate the weight of the processor graph as follows:

$$O = \sum_{i=1}^{n} W_i$$

O is the objective function for this heuristic and the main goal of the algorithm is to maximize this objective function.

We present the pseudo code for our heuristics below:

4.1 IFFD Algorithm

```
AllocateProcessor()
1. For each task $t_i$ based on its priority $P_i$ (highest priority first)
    1.1. Check its neighbors in processor constraint graph $G_0$ (not the same as $G_p$)
    1.2. For each neighbor $t_j$ based on its priority $P_j$ (lowest priority first)
        1.2.1. Initialize constraint count to zero
        1.2.2. For each constraint $R_{jk}$ for that task $t_j$
            1.2.2.1. If there is an edge $\langle t_i, t_j\rangle$ in the constraint graph $G_k$ increase the constraint count
        1.2.3. If all the constraints are satisfied and task $t_j$ is not optimized, then task $t_j$ is the best match.
        1.2.4. Else increment j, goto 1.2.1
        1.2.5. Allocate task $t_i$, $t_j$ to the same cluster using pair-wise exchange.
        1.2.6. If allocation is successful, mark $t_i$ and $t_j$ as optimized, increment i, and goto 1.1.
        1.2.7. Else increment j.
    1.3. For each neighbor $t_j$ in decreasing order of constraints satisfied
        1.3.1. Allocate task to the same cluster using pair-wise exchange.
```

1.3.2. If allocation is successful, increment i, goto 1.1.
1.3.3. Else increment j.

Let us elaborate on the pair-wise exchange. Suppose one wants to get task x_a and y_b to the same cluster. Let the processor requirements of the un-optimized tasks in the two clusters be represented by set $X = \{ x_1, x_2 \ldots\ldots x_k \}$ and $Y = \{y_1, y_2 \ldots\ldots y_m \}$, where

$$\sum_{i=1}^{k} x_i = p$$

$$\sum_{i=1}^{m} y_j = p$$

and p is the number of processors in cluster. When we perform pair-wise exchange, the tasks in the two clusters under consideration are sorted in increasing order according to their processor requirement.

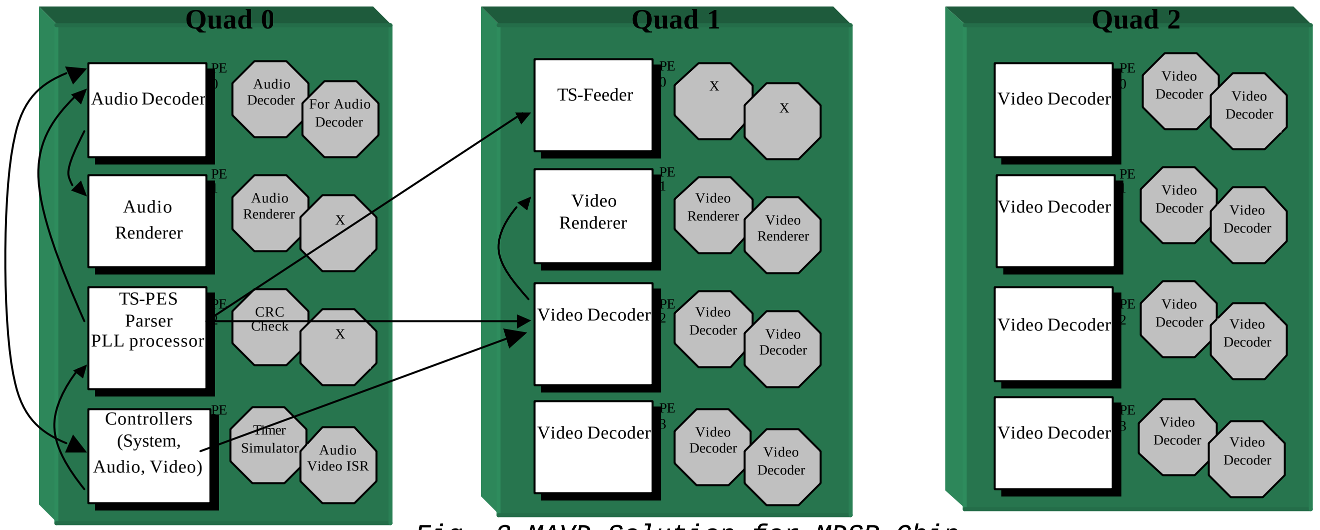


Fig. 3 MAVD Solution for MDSP Chip

4.2 Pair-wise Exchange Algorithm for IFFD

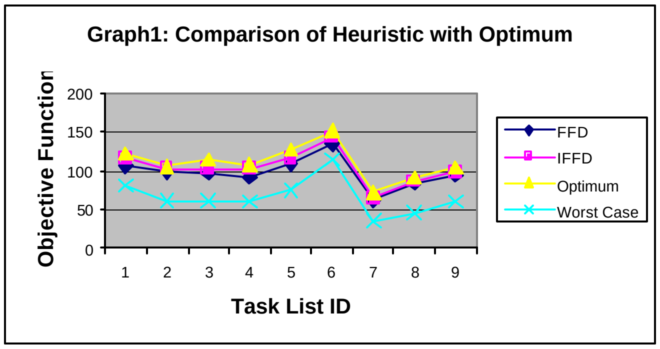
Allocation of task to the same cluster (pair-wise exchange algorithm)

```
1. Check if the two tasks $x_a$, $y_b$ can co-exist in the same cluster, by checking their processor requirement. If not return false
2. For each sum s from 1 to p
    2.1. Check if a subset sum s for two elements exists in $X$
    2.2. Check if the same subset sum s for two elements exists in $Y$
    2.3. If found swap
3. Calculate the weight of the processor graph. If the value of the processor graph is reduced then revert the changes made and return false, else return true.
```

Lemma 1

The approximation factor of this heuristic is not less than that obtained by FFD heuristics.

Proof



The initial mapping is obtained using FFD. During the pair-wise mapping after each exchange we re-calculate the value of the objective function and if this value is less than the value obtained we revert the changes. In this way we make sure that we don't degrade the approximation obtained by the FFD heuristic □

Lemma 2

Given an input set S of integer and another integer x it takes O(nlogn) to determine whether there exists two elements in S whose sum is exactly x.

Proof

Sort the input set S using a $O(n\log n)$ comparison sort algorithm. For each element y_i in the set S, search for z_i where $z_i = x\text{-}y_i$. The search can be done using binary search on sorted elements which takes $O(logn)$ time. So the total time is $O(nlogn) + O(logn) = O(nlogn)$ time.

Therefore the total time required for pairwise exchange is $O(pnlogn)$, where n is the max task on one of the two clusters in consideration and p is the number of processors in the cluster □

Till now, we have not discussed the case where the processor requirement of the task exceeds the processor availability in the cluster. The solution is to split the tasks and allocate it on separate clusters. Since the task is split it cannot be optimized easily. One solution we have implemented is that if the task is split into z_1, z_2… z_n units then we try to optimize it on z_i, which is the largest fragment of the task. The remaining fragments will always remain un-optimized and can be squeezed into any available free segment. The MAVD system is a typical example where the requirement of 6 processors by the video decoder cannot be satisfied by one cluster. In this case we have $z_1 = 4$ and $z_2 = 2$. We considered z_1 as the main task and optimize on z_1. The fragment z_2 can be squeezed into any available processor. The other solution will be to consider z_1 and z_2 as separate tasks and optimize the mapping on both the tasks. The communication constraint will have the highest priority for z_1 and z_2, in case there is heavy intra-task communication.

5.0 Experimental Results

Graph 1 compares the object function value obtained for FDD, IFFD, the Optimum Algorithm and the Worst Case Algorithm. For testing the heuristics, we generated synthetic tasks graphs.

The resource requirements for the set of tasks were generated using the pseudorandom function such that the sum of resources was always less than or equal to the sum of the resources available on the MDSP chip. We have modeled our testing on a five cluster MDSP chip, with each cluster consisting of four PEs, eight DSEs, and 64Kbytes of data memory. We calculate the objective function for mapping generated by our heuristics.

We also calculated the objective function for the initial state obtained using FFD algorithm. In most of the cases our algorithm generated better results than FFD algorithm. For test results, higher value indicates better results.

The objective function for an optimum mapping scheme was calculated and was found to be the best. In addition to this the worst-case objective function was also calculated. The result for the tests conducted is depicted in Graph 1.

The objective function for an optimum mapping scheme was calculated and was found to be the best. In addition to this the worst-case objective function was also calculated.

5.1 MAVD System

In the previous section we had discussed the MAVD system developed by Cradle Technologies. We mapped the system using our IFFD algorithm and obtained the solution using our heuristics. It is shown in Fig. 3.

In the MAVD system, the Video Decoder and Video Renderer has more communication, so they are placed within the same cluster. Similarly, Audio Decoder and Audio Renderer are placed in the same cluster. There is more communication between TS-PES Parser task and Audio Decoder task, so they have to be placed in the same cluster. Similarly, the System Controller has more communication with Audio Decoder due to higher rater of Audio ISR than Video ISR and needs to be placed in the same cluster.

We compared this mapping obtained by our algorithm and the mapping done manually by experienced system designers from Cradle over the span of two years and found it to be same. The design had gone through multiple iterations before it was finalized, while we obtained it using our tool within the first iteration.

6.0 Concluding Remarks

We have presented a mapping heuristic in this paper. We conducted some tests using synthetic task graphs and a real system such as MAVD. The confidence levels for development of such heuristics was boosted when it was used on some existing systems and the results were comparable to those obtained manually and better than those obtained using plain FFD or BF heuristic. With more and more clustered multi-core processors being introduced in the commodity processor market to cater to the increasing performance needs of the high-end media and networking applications, efficient heuristic to design such systems will facilitate the development thereby making these processors acceptable by general programming community.

7.0 References

[1] Chretienne, Ph., Coffman, E.G., Jr., Lenstra, J.K., and Liu, Z. (Editors), "Scheduling Theory and Its Applications", *Wiley, Chichester, England (1995).*

[2] "3400 Hardware Architecture Reference", *Cradle Technologies, www.cradle.com.*

[3] ISO/IEC 13818: "Generic coding of moving pictures and associated audio (MPEG-2)"

[4] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, and K. C. Sevcik. "Theory and Practice in Parallel Job Scheduling." *Lecture Notes in Computer Science, 1291:1--34, 1997.*

[5] Renaud Lepère, Denis Trystram, Gerhard J. Woeginger: "Approximation Algorithms for Scheduling Malleable Tasks Under Precedence Constraints." *Int. J. Found. Comput. Sci. 13(4): 613-627 (2002)*

[6] H. Shachnai, J. Turek, "Multiresource Malleable Task Scheduling", *in IPL, vol. 70, 1999, pp. 47—52*

[7] V. Chaudhary and J. K. Aggarwal, "A generalized scheme for mapping parallel algorithms", *in IEEE Trans. on Parallel and Distributed Systems, Mar '93, pp. 328 - 346.*

[8] Garey, Johnson, "Computers and Intractability: A guide to the theory of NP-Completeness", *W.H Freeman and Company (1979)*

[9] E. G. Coffman, M. R. Garey, D. S. Johnson "Approximation algorithm for Bin Packing: A Survey, Appears in Approximation Algorithms for NP-Hard Problems", *D. Hochbaum(ed) PWS Publishing(1996)*

[10] K.Maruyama, S. K. Chang, and D. T. Tang. "A general packing algorithm for multidimensional resource requirements." *Intl. J.of Comput. and Inf. Sci., 6(2):131–149, May 1976.*

[11] W. Leinberger, G. Karypis, and V. Kumar. "Multi-Capacity Bin Packing Algorithms with Applications to Job Scheduling under Multiple Constraints." *In Proceedings of the 1999 International Conference On Parallel Processing, 1999.*

[12] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey and R. L. Graham. "Worst Case bounds for simple one dimensional packing algorithms." *SIAM Journal on computing, 3:299—325,1974.*

[13] H. Rock, G. Schmidt; "Machine aggregation heuristics in shop scheduling" *Math Oper Res. 45 (1983) 303-314*

[14] Anand Srivastav, Peter Stangier: "Tight Approximations for Resource Constrained Scheduling and Bin Packing." *Discrete Applied Mathematics 79(1-3): 223-245 (1997)*

[15] Y.-K. Kwok, I. Ahmad. "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors." *IEEE Trans. on Parallel and Distributed Systems, vol.7, pages 506--521, March 1996*

[16] A Feldman, M Kao, J Sgall, S Teng, Optimal online scheduling of parallel jobs with dependencies, *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing, p.642-651, May 16-18, 1993*

[17] A Feldmann, J Sgall ,S Teng Dynamic scheduling on parallel machines, *Proceedings of the 32nd annual symposium on Foundations of computer science, pp 111 - 120, 1991*

[18] H. Shachnai, J. Turek, " Multiresource Malleable Task Scheduling", *in IPL, vol. 70, 1999, pp. 47—52*

[19] Klaus Jansen, Lorant Porkolab, "Linear-time approximation schemes for scheduling malleable parallel tasks." *Symposium on Discrete Algorithms, Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms, pp. 490 - 498, 1999*