A Static Scheduling Heuristic for Heterogeneous Processors

Hyunok Oh        Soonhoi Ha *

17 SCHEDULING AND LOAD BALANCING

This paper presented a static scheduling heuristic called best-imaginary-level (BIL) scheduling for heterogeneous processors. The input graph is an acyclic precedence graph, where a node has different execution times on different processors. The static level of a node, BIL, incorporates the effect of interprocessor communication (IPC) overhead and processor heterogeneity. The proposed scheduling technique is proven to produce the optimal scheduling result if the topology of the input task graph is linear. The proposed scheduling is expected to be applicable to the large span of target architectures from the network computing to the hardware/software codesign.

keywords: multiprocessor scheduling, heterogeneous scheduling, static scheduling, best imaginary level, BIL

1 Introduction

Thanks to the rapid development of networks, the distributed systems, which consist of various workstations, main computers, and even super computers, become a promising workhorse to increase the computing power. These systems are usually heterogeneous systems with significant overhead of interprocessor communication (IPC).

Other examples of heterogeneous systems can be found in embedded systems. They are application specific systems which contain hardware and/or software tailored for a particular task. General purpose processors, special purpose processors, programmable digital signal processors, and ASICs are among the many components of these systems. In these systems, heterogeneous processors are tightly coupled with low IPC overhead but with heavy resource constraints.

Therefore, the schedulers for such heterogeneous systems need to account for IPC overhead and processor heterogeneity: a task takes different execution times on different processors. A rich history of multiprocessor scheduling has been built for homogeneous processors to illustrate that the scheduling problem is very difficult [2]. Since a homogeneous system is a special case of a heterogeneous system, the scheduling problem for heterogeneous processors is much more difficult. In this paper, we have developed a nonpreemptive static scheduling heuristic called best-imaginary-level (BIL) scheduling.

The proposed scheduling takes an acyclic precedence graph (APG) as an input task graph. A node of an APG represents a task to be executed on a processor and a directed arc indicates the precedence relation between two nodes. Each arc is associated with a number that specifies the amount of IPC overhead. We assume that the execution time of a node $N_i$ on a processor $P_j$, $E(N_i, P_j)$, is known at compile-time.

*They are with the Department of Computer Engineering, Seoul National University, Seoul, 151-742, Korea; fax number: 82-2-8867389 e-mail: oho@cc2.snu.ac.kr sha@cc2.snu.ac.kr
This research is supported by the S.N.U. Research Fund
for each node-processor pair. If node \( N_i \) cannot be executed on processor \( P_j \), \( E(N_i, P_j) \) is infinite. This occurs when processor \( P_j \) has no resource nor instruction for node \( N_i \). An example APG and its node execution-time table is shown in figure 1. We also assume that the target architecture has a dedicated hardware for IPC so that interprocessor communication time can be overlapped with computation time in a schedule.

The scheduling objective in this paper is to minimize the scheduling length or makespan of the input APG. Since this scheduling problem is NP-hard in the strong sense [3], we will rely on heuristics. As an existing scheduling heuristic for heterogeneous scheduling problem, we will consider the General Dynamic Level (GDL) scheduling technique developed by G.C.Sih and E.A.Lee [1]. We will show that the proposed technique is more algorithmically appealing to generate an optimal schedule for a special class of APG. Moreover, we obtain about 20% performance improvement over GDL with randomly generated APGs.

Both GDL and the proposed BIL scheduler are based on the list scheduling idea [4]. Each node is assigned a priority. We call a node runnable when its ancestor nodes were scheduled already. The list scheduling schedules the runnable nodes in the decreasing order of priority. The variants of list scheduling techniques differ in terms of how to assign priorities to the nodes and on which processor a selected node is to be scheduled [5]. In the next section, we will briefly review the GDL scheduling technique to show its limitation. we will explain the proposed BIL scheduling technique in section 3. Experimental results and conclusions will be followed in section 4 and 5 respectively.

2 GDL Scheduling Technique

The schedule length can not be less than the maximum length, or critical path length, of a node to a terminal node. Therefore, to minimize the schedule length, it is intuitive to assign the highest priority to a node of the longest critical path length. The critical path length of a node is the sum of execution times of nodes on the critical path and the IPC overheads incurred if these nodes are assigned to the different processors. Since the IPC overheads are not known before the descendants nodes are scheduled, a typical
list scheduling technique considers only the sum of execution times on the critical path to determine the priority of a node. We call it the static level, $SL(N_i)$, of a node. The static level of a node becomes the critical path length if all nodes on the critical path are scheduled to the same processor.

In a heterogeneous system, however, a node has different execution times on different processors. If we choose the smallest execution time of a node in the static level computation, it may not be possible to assign the node to the same processor as its ancestor. In this case, ignoring the IPC overhead is problematic specially for such a heterogeneous system as a distributed system. Therefore, the GDL scheduler defines the assumed execution time of node $N_i$, denoted $E^*(N_i)$, as the median execution time of node $N_i$ over all the processors. If the median execution time is infinite, the largest finite execution time is substituted.

The GDL scheduler considers the effects of IPC overhead by adjusting the level when the node becomes runnable. We define $DA(N_i, P_j)$ to be the earliest time that all data required by node $N_i$ are available at B processor $P_j$, where the IPC overhead is accounted for. If we further define $TF(P_j)$ as the time that the last node assigned to the jth processor finishes execution, the node $N_i$ can not be scheduled before $T(N_i, P_j)$, which is the maximum of $DA(N_i, P_j)$ and $TF(P_j)$, on processor $P_j$. To account for the varying processing speeds, they also defined

$$\Delta(N_i, P_j) = E^*(N_i) - E(N_i, P_j).$$

A large positive $\Delta(N_i, P_j)$ indicates that processor $P_j$ executes node $N_i$ more rapidly than most processors, while a large negative $\Delta(N_i, P_j)$ indicates the reverse. By incorporating these terms, their first extended the static level to the dynamic level $DL_1(N_i, P_j)$ as follows:

$$DL_1(N_i, P_j) = SL(N_i) - T(N_i, P_j) + \Delta(N_i, P_j).$$

This level gives a higher priority to a node with regard to processor $P_j$ the higher static level it has or the earlier it can be scheduled or the faster it can be executed. Although $DL_1(N_i, P_j)$ indicates how well node $N_i$ is matched with processor $P_j$, it fails to consider how well the descendants of $N_i$ are matched with $P_j$. If a descendant of node $N_i$ can not be scheduled on $P_j$ due to the resource constraints, the IPC overhead between node $N_i$ and its descendant should be considered to estimate the cost of assigning $N_i$ to processor $P_j$. From this observation, GDL scheduler selects a descendant of a node with the largest IPC overhead, $D(N_i)$ and defines another term $F[N_i, D(N_i), P_j]$ to indicate how quickly $D(N_i)$ can be completed on any other processor if $N_i$ is executed on $P_j$. Then, the effect of the descendants on the level of node $N_i$ becomes

$$DC(N_i, P_j) = E^*[D(N_i)] - \min(E[D(N_i), P_j], F[N_i, D(N_i), P_j]),$$

which roughly indicates how well the ”most expensive” descendant of node $N_i$ matches with processor $P_j$, if $N_i$ is scheduled on $P_j$. Incorporating the descendant consideration term modifies the level of a node on processor $P_j$ as follows:

$$DL_2(N_i, P_j) = DL_1(N_i, P_j) + DC(N_i, P_j).$$

In addition to the descendant consideration effect, a heterogeneous processing environment also introduces a cost associated with a node if it is unable to be executed on a certain processor. To characterize this resource scarcity cost, they selects the optimal processor $P_j^*$ to maximize $DL_2(N_i, P_j)$ and the second best processor. Then, the ”generalized” dynamic level is defined as follows:

$$GDL(N_i, P) = DL_2(N_i, P_j^*) + C(N_i).$$

The second term is the increase of the dynamic level if the node is forced to be scheduled on the second optimal processor. Now the GDL scheduling technique selects the runnable node of highest priority, or highest GDL. Note that when a node is selected, the optimal processor is already determined.
Figure 2: An example to show the effects of descendant consideration

<table>
<thead>
<tr>
<th></th>
<th>P0</th>
<th>P1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 3: scheduling result from the GDL scheduler of figure 2.

The GDL scheduling inherits the same weakness of the conventional list scheduling that the global effect of the scheduling decision is not measured properly. As an effort of improvement, the GDL scheduler quantifies the scheduling effects on the descendants of a node. However, considering the first descendants is not enough to measure the global effect. Consider an example shown in figure 2. The scheduling result from the GDL heuristic is represented in figure 3 with a Gantt chart that represents on which processor and at which time each node is scheduled. When node A is scheduled, the GDL fails to consider the effect of the processor selection on node C. Therefore, node A is scheduled on processor P0. As a result, it suffers from the huge IPC overhead between A and B since node B should be scheduled to processor P1 considering node C.

In the following sections, the proposed heuristic will be explained how it overcomes the drawbacks of the list scheduler.

3 The Proposed BIL Scheduler

In a homogeneous system, the static level of a node can be regarded as the critical path length from the node to a terminal node with an ideal, but usually unrealistic, assumption that all nodes on the critical path are scheduled back-to-back on the same processor. We regard the static level of a node as a special form of the best imaginary level (BIL) of a node when the system is homogeneous. In general, the BIL of a node is based on the following assumption.

Assumption 1 All descendant nodes can be scheduled at the best times.

In most cases, this assumption will fail since more than one nodes want to be scheduled on the same processor at the same time. Therefore, we use the term best imaginary to indicate that it is not a real situation.

3.1 Best Imaginary Level Computation

Now, we define the general form of the BIL of a node as follows.
Definition 1

\[ BIL(N_i, P_j) = E(N_i, P_j) + \max_{d_i \in D(N_i)} \left[ \min(BIL(d_i, P_j), \min_{k \neq j}(BIL(d_i, P_k) + d(N_i, d_i))) \right] \]

where \( d(N_i, D(N_i)) \) means the amount of IPC overhead between \( N_i \) and the descendant \( D(N_i) \).

The BIL of node \( N_i \) indicates the critical path length including the IPC overhead assuming the node is scheduled on processor \( P_j \). Since it considers the IPC overhead in its computation, the BIL of a node can be thought as the global information of all descendant nodes under the optimistic assumption. Therefore, the BIL scheduler shows a drastic improvement with the example of figure 1 compared with the GDL scheduler which fails to account for the global information. The scheduling result is shown in figure 4. Node A is now scheduled on processor P1 since the scheduler considers the effect of the far descendant C reflected in the BIL computation of node A. In fact, the BIL scheduler produces the optimal scheduling result if the APG is linear.

Theorem 1 The BIL scheduler produces the optimal scheduling result if the APG is linear.

[Proof] We first prove by induction that the BIL\((N_i, P_j)\) indicates the shortest finish time starting from node \( N_i \) to the terminal node assuming that node \( N_i \) is scheduled on processor \( P_j \).

i) In case that the number of node is 1
\( BIL(N_1, P_j) = E(N_1, P_j) \) by definition. So BIL trivially indicates the shortest finish time.

ii) Assume that \( BIL(N_2, P_j) \) indicates the shortest finish time from node \( N_2 \) to the terminal node.

iii) We append a node, \( N_1 \), in front of the linear graph.
If \( E(N_1, P_j) + \min_k (BIL(N_2, P_k) + d(N_1, N_2)) < E(N_1, P_j) + BIL(N_2, P_j) \) then the shortest finish time of node \( N_1 \) executed on processor \( P_j \) occurs when node \( N_2 \) is scheduled on processor \( P_j \) which minimize \( BIL(N_2, P_k) + d(N_1, N_2) \) term.
If \( E(N_1, P_j) + \min_k (BIL(N_2, P_k) + d(N_1, N_2)) \geq E(N_1, P_j) + BIL(N_2, P_j) \) then the shortest finish time of node \( N_1 \) executed on processor \( P_j \) occurs when node \( N_2 \) is also scheduled on processor \( P_j \). Thus, we proved that the \( BIL(N_i, P_j) \) is the shortest finish time from node \( N_i \) to the terminal node.

Now, we prove that the BIL scheduler produces the optimal schedule if the APG is linear. The proof is trivial since the optimal schedule length is nothing but the minimum value of the BIL values of the starting node, \( N_1 \). In other words, \( \min_j BIL(N_1, P_j) \) is the optimal scheduling length. Since the \( BIL(N_1, P_j) \) contains the information on which processor the child node is to be scheduled, the BIL scheduler completes the schedule once it determines the minimum value of the BIL value of node \( N_1 \).

Therefore the BIL Scheduler produces the optimal scheduling result if the APG is linear. Q.E.D.

3.2 Node Selection

As the scheduling proceeds, we adjust the level of a node \( N_i \) on processor \( P_j \) to indicate the best imaginary makespan, \( BIM \). \( BIM \) is defined as follows:
\[ BIM(N_i, P_j) = T(N_i, P_j) + BIL(N_i, P_j) \]
Note that a runnable node has \( N \) different \( BIM \) values, one for each processor, if the total number of processors is \( N \).
Among the runnable nodes, we select a node to be scheduled based on a pessimistic assumption in order to decrease the performance penalty in case our scheduling decision is far from optimal. Assuming that the number of runnable nodes at a scheduling stage is $k$, the proposed strategy is described below.

i) In case of $k < N$
A node is assumed to be selected at the $k$-th turn. We sort the BIMs of a node in the ascending order and compare the $k$-th BIM with those of other nodes to select the node of higher BIM value. If the $k$-th BIM value of a node is infinite, the largest finite BIM value is used for comparison since it is guaranteed that the node will not be scheduled on the processors associated with infinite BIM values.

ii) In case of $N \leq k$
Since a node has only $N$ BIM values, we compare the largest finite BIM values of runnable nodes.

In case more than one nodes have the same BIM values to be selected, we adopt a tie breaking policy in a recursive form. If the $k$-th BIM values of nodes are the same, then we compare the $(k - 1)$-th BIMs of nodes that have the same $k$-th BIM.

### 3.3 Processor Selection

The next step is to determine the optimal processor. Even though we are apt to select the processor associated with the smallest BIM value, this selection scheme may not be optimal since the BIL of a node is pessimistic assuming that the IPC overhead is visible. If the number of runnable nodes is large, the IPC overhead between a node and its child is likely to be hidden by other runnable nodes. Therefore, in case the the parallelism of the APG is high, the execution time becomes the more important factor than the IPC overhead. Consider an example of figure 5. At the first step, node A is selected and scheduled on processor P0 since the BIM value is the smaller on P0. But this decision is not good as shown in figure 6.

When node A is selected, the number of runnable node is 4 and the number of processors is 2. Therefore, we may expect that on the average two nodes will be scheduled on each processor before its child node is scheduled, assuming that the BIM values of the runnable nodes are not significantly different. It means that the IPC overhead between nodes A and B is likely to be hidden while the BIL does not assume that. In other words, the BIL scheduler is pessimistic about the graph parallelism. Now, we have to compensate this factor to define the revised BIM as follows.
amp = min\left[ \frac{k}{N} - 1, 0 \right]

BIM^*(N_i, P_j) = T(N_i, P_j) + BIL(N_i, P_j) + E(N_i, P_j) \times amp

Using this revised BIM value, we obtain the better result as shown in figure 7.

If more than one processors have the same revised BIM value, the BIL scheduler adopts another tie-breaking policy. We select the processor that makes the sum of the BIM values of other nodes on the processor maximum. It is because the other nodes had better not be scheduled on the processor if their BIM values are large.

4 Experiments

To estimate the performance of the BIL scheduler, we used a random APG generator with various numbers of nodes and processors as well as graph parallelism. We performed 100 experiments with 30, 50, 100, 150, 200, 250, 300, 350, 400 and 500 nodes on 2, 3, 5, 7, 10, 12, 15, 20, 25 and 30 processors. We repeated 10 times with different seeds for random number generation for each pair of nodes and processors and averaged the results. We showed the performance improvement over the GDL in figure 8 by varying the ratio of the IPC overhead and the average execution times of nodes by 0.5, 1 and 2. Let \( \bar{C} \) be the mean IPC overhead and \( \bar{E} \) be the average node execution time. For instance, \( \bar{C} = 0.5 \times \bar{E} \) means that the IPC overhead is a half of the average execution time.

The overall performance comparison is shown in table 1. We observed that that the performance of the BIL scheduler becomes dominant as the number of processors is greater than 2. We conjecture the reason of performance differences two-folds. First, as the number of processors grows, the effect of graph parallelism would be decreased, so that the BIL scheduler consider the IPC more effectively than the GDL scheduler. Second, the GDL examines the two processors per node to consider the resource scarcity as explained in the section 2.

The time complexity of this technique is \( O(n \times (pn + p)) \) in the BIL computation and \( O(n(pn \log p + np + pn + p)) \) in the revised BIM computation to make \( O(n^2 p \log p) \) in total, where \( n \) is the number of nodes and \( p \) is the number of processors. It is smaller than the complexity of GDL. But, our experiments revealed that it takes about the same time with the GDL scheduler because the coefficients of the BIL is larger than that of the GDL while the order is smaller.
Figure 8: Percentage improvement of the BIL scheduler over the GDL scheduler. (a) $\tilde{C} = 0.5 \times \tilde{E}$ (b) $\tilde{C} = \tilde{E}$ (c) $\tilde{C} = 2 \times \tilde{E}$.

Table 1: Percentage improvement in speed-up of the BIL scheduler over the GDL scheduler

<table>
<thead>
<tr>
<th>Speed-up</th>
<th>Mean</th>
<th>Std. deviation</th>
<th>Minimum</th>
<th>Median</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>117%</td>
<td>12.52%</td>
<td>78.46%</td>
<td>116.50%</td>
<td>172.17%</td>
</tr>
</tbody>
</table>

5 CONCLUSION

We have proposed a non-preemptive static scheduling heuristic called Best-Imaginary-Level (BIL) scheduling for heterogeneous processors. The input graph is an acyclic precedence graph, where a node has different execution times on different processors. The BIL of a node is the critical path length from the node to a terminal node including the IPC overhead as well as the execution times of the nodes on the critical path.

The proposed scheduling technique is proven to produce the optimal scheduling result if the topology of the input task graph is linear. The performance of the BIL scheduling is compared with an existing technique called the general dynamic level (GDL) scheduling with various classes of randomly generated input graphs, resulting in about 20% performance improvement. The time complexity of the scheduling is $O(n^2p \log p)$, where $n$ is the number of nodes, and $p$ is the number of processors.

The proposed scheduling is expected to be applicable to the large span of target architectures from the network computing to the hardware/software codesign. For more general applicability, the heuristic needs to be extended to consider the effects of resource limitation. We are currently estimating the performance bounds for any general APG.

References


