Part I

COMPUTER SCIENCE
MODELING OF SUPERSCALAR INSTRUCTION SCHEDULING AND ANALYSIS OF A HEURISTIC SCHEDULING ALGORITHM

HONG CHICH CHOU and CHUNG PING CHUNG

Institute of Computer Science and Information Engineering
National Chiao-Tung University
Hsinchu 30050, Taiwan, R.O.C.

Abstract.

The problem of superscalar instruction scheduling is studied and an analysis of a heuristic scheduling algorithm is presented. First, a superscalar architecture is characterized by \( k \), the number of types of functional units employed, \( m_i \), the number of type \( i \) functional units, \( P_{ij} \), the \( j \)th functional unit of type \( i \), and \( z \), the maximal number of delay cycles incurred by the execution of instructions. A program trace to be scheduled is modeled by a directed acyclic graph with delay on precedence relations. These two models reflect most of the flavor of the superscalar instruction scheduling problem. A heuristic scheduling algorithm called the ECG-algorithm is designed by compiling two scheduling guidelines. The performance of the ECG-algorithm is evaluated through worst-case analysis. Letting \( W_{ECG} \) denote the length of an ECG-schedule and \( W_{opt} \) the length of an optimal schedule, we established the bound \( W_{ECG}/W_{opt} \leq k + 1 - 2/\max \{m_i\}(z + 1) \), which is smaller than other known bounds.

Classification CR: D.4.1, I.3.1, I.2.8, B.2.2

Keywords: Scheduling, Parallel processing, Heuristic methods, Worst-case analysis.

1. Introduction.

In contrast to vector processors, which are used to execute scientific programs, superscalar processors are microprocessors which attempt to improve the execution rate of non-scientific programs by executing, on average, more than one instruction per clock cycle. Examples of superscalar architectures include the IBM RS/6000, Intel i860, [1, 3], and DEC Alpha. The success of superscalar machines depends not only on the vast hardware resources they provide, but even more importantly, on how efficiently these resources can be used. Efficient use of resources and reduced program execution time may be achieved through instruction scheduling. The
objective of superscalar instruction scheduling is to rearrange instructions in such a way that they are executed by hardware in an optimal (or near optimal) order (that is, so that execution time is minimized). The problem of finding an optimal order is intractable, however [10].

Superscalar instructions are characterized by their types and delay cycles. The instructions may be classified into several types, such as fixed-point instructions and floating-point instructions. Each type can only be executed on corresponding type of functional unit. Though functional units are pipelined, not all instructions can produce results in one cycle. For example, a load instruction needs an additional cycle to deliver the fetched data to the waiting instructions. In this case, the load instruction is said to have one delay cycle.

Instruction scheduling can be done by hardware or by software. The scheduling of instructions by hardware was pioneered by Tomasulo [5], who designed a hardware instruction issuing mechanism called reservation stations for the IBM 360 model 91. Recent research can be found in [4]. To be fully functional, however, hardware scheduling must solve several problems, including artificial dependencies caused by register-file limitations, conditional branches, and imprecise interrupts. Solving all these problems requires hardware that is complex, costly, and, more seriously, slow.

Because of the drawbacks of hardware scheduling, instruction scheduling is mostly done by software, as in the case of the VLIW, IBM RS/6000, and Intel i860. The VLIW machine [7] is an extreme case which relies in a complex way on software instruction scheduling to keep functional units busy while avoiding any data hazards [2]. In addition to identifying instructions that can be executed concurrently, the scheduling software also tries to optimize delay slots. Since software exposes program parallelism to processors, the architecture can be greatly simplified by assuming that the incoming instructions in each cycle can be executed simultaneously, as in the approach of the IBM RS/6000, or by setting one bit in the instructions to notify the processor to execute the next instructions in parallel, as in the approach of the Intel i860. A simple hardware design often implies a fast clock rate.

Scheduling algorithms are generally evaluated through benchmarking [8, 9]. The execution times of the unscheduled and scheduled codes are compared as a performance criterion for the scheduling algorithms. However, benchmarking suffers from a number of drawbacks: (1) it is costly, since it requires a simulator and a compiler to generate code; (2) it is time-consuming; (3) it gives only statistical results and it fails to analyze the nature of an algorithm; and (4) it is a form of post-analysis, which can be done only after a processor has already been designed. Thus in this paper, instead of using benchmarking, we evaluate a scheduling algorithm (the algorithm is given below) from a mathematical point of view, using worst-case performance analysis to reveal the performance limit of the algorithm.
2. Formal specifications.

2.1. Abstract models.

Since different superscalar machines have diverse types of organizations, to keep
our study universal and independent of particular machines, we define an abstract
machine model and a model for a trace of code to be scheduled. As can be seen from
currently available superscalar architectures and the demands of instruction sched-
uling, a superscalar architecture can be denoted by a 4-tuple \((k, M, \mathcal{P}, z)\), where:

- \(k \geq 1\) denotes the number of different types of processors employed in the
  superscalar architecture.
- \(M = (m_1, m_2, \ldots, m_k)\), where \(m_i\) denotes the number of type \(i\) processors, for
  \(1 \leq i \leq k\). The instruction set is also categorized into \(k\) types. Type \(i\) instructions
  can only be executed on any of the \(m_i\) type \(i\) processors.
- \(\mathcal{P} = \{P_{ij} \mid 1 \leq i \leq k, 1 \leq j \leq m_i\}\) denotes the set of processors. \(P_{ij}\) denotes the \(j\)th
  processor of type \(i\).
- \(z \geq 0\) denotes the maximum number of pipeline delay cycles required to execute
  any instruction.

The above machine model is generalized enough to encompass a wide range of
superscalar architectures. For example, for the IBM RS/6000, \(k = 4, m_1 = m_2 =
\ldots = m_4 = m_3 = 1\), and \(z = 6\) (floating-point compare instruction delay).

By convention, an instruction is said to have no delay if its execution is completed
within one cycle and to have \(x\) cycles of delay if it needs \(x\) additional cycles to be
completed.

We formally denote a program trace to be scheduled by a 4-tuple \((U, <, P_f, D)\),
where:

- \(U = \{I_1, I_2, \ldots, I_t\}\), is the aggregation of the instructions to be scheduled
- \(<\) is a transitive binary relation defined on \(U\) that specifies the precedence
  relationships between instructions.
- \(P_f\) is a function, \(P_f : U \rightarrow \{1, 2, \ldots, k\}\), specifying the type of processor on which
  an instruction can be executed.
- \(D\) is a function, \(D : U \rightarrow \{0, 1, \ldots, z\}\), specifying the number of delay cycles of each
  instruction.

precedence relations originate from true data dependencies, procedural depend-
encies, antidependencies, and output dependencies. If \(I_i < I_j\), this means that \(I_i\)
must be executed before \(I_j\). It is unnecessary to distinguish among these dependencies
here, since they impose the same effect on instruction execution. Dependence and
precedence are used interchangeably in the following context.

It would be convenient to represent each \((U, <, P_f, D)\) instance by a directed
acyclic graph (DAG). Figure 1 shows an instructions DAG with three types and
a maximum delay of two cycles. Nodes (Circles) in Figure 1 denote instructions. If
\(I_i < I_j\), then there is a directed edge from node \(I_i\) to node \(I_j\) in the graph (note that
transitive edges are not shown in the figure). The number behind the slash in each node indicates the type of that instruction, and the number besides each edge denotes the number of delay cycles required by the predecessor. If $I_i < I_j$, then $I_j(I_i)$ is a successor (predecessor) of $I_i(I_j)$. If there is no $I'$ such that $I_i < I' < I_j$, then $I_j(I_i)$ is said to be an immediate successor (predecessor) of $I_i(I_j)$. $P(I_i)$ is used to denote the set of predecessors of $I_i$, and $S(I_i)$ the set of successors of $I_i$ (not necessarily immediate in both sets).

A function $\lambda : U \rightarrow \{1, 2, \ldots \}$ is a feasible schedule for a given $(U, <, P_f, D)$ on a given $(k, M, P, z)$, if $\lambda()$ meets the following conditions:

1. $\forall I_i, I_j \in U$, if $I_i < I_j$, then $\lambda(I_i) + D(I_i) < \lambda(I_j)$.
2. $\forall t \geq 1 \land i \leq k$, $|\{I \in U \mid \lambda(I) = t \text{ and } P_f(I) = i\}| \leq m_i$.

Condition 1 requires that a schedule should observe the precedence relationships; condition 2 requires that a schedule should not use more processors than available. When an instruction $I_i$ is said to be scheduled in time slot $t_i$, this means that $\lambda(I_i) = t_i$. None of the successors of $I_i$ can be issued until $\lambda(I_i) + D(I_i) + 1$. We define $\asth(I_i) + D(I_i)$ to be the completion time of $I_i$ and use the function $\text{Len}(I_i) = D(I_i) + 1$ to denote the length of an instruction. The time slots $(\asth(I_i), \lambda(I_i) + D(I_i))$ are said to be the scope of $I_i$. The length of a scope is also the length of that instruction. $\text{Len}()$ is also used to denote the length of a chain. A chain $h = \langle I_1, I_2, \ldots, I_x \rangle$ is a sequence of dependent

Figure 1. A sample instruction DAG.
instructions, such that $I_i$ is an immediate predecessor of $I_{i+1}$. The length of a chain is the sum of the lengths of all instructions in the chain. The length of a feasible schedule is defined to be $\max \{ \lambda(I) + D(I) \}, \forall I \in U$.

3. Heuristic algorithm.

Since it is not likely that an essential fast algorithm could be found for an NP-complete problem, we propose a heuristic scheduling algorithm called the ECG-algorithm. The algorithm is a List-scheduling-like algorithm. Each instruction is given a label to reflect the scheduling priority of that instruction. The scheduling discipline is as follows: Whenever a processor becomes idle, the instruction with the largest label among the instructions ready to be issued is selected to be executed on the idle processor.

Let $\alpha(I_i)$ denote the label of $I_i$. The ECG-algorithm is designed by the following two scheduling guidelines. For a given $(U, \prec, Pf, D)$, we define two relations among instructions, as follows:

**Definition 1.** Dominance relation on instructions. For $I_i, I_j \in U$ and $Pf(I_i) = Pf(I_j)$, $I_i$ is said to dominate $I_j$, denoted as $I_i \dom I_j$, if $S(I_i) \geq S(I_j)$ and $P(I_i) \subseteq P(I_j)$ and $\sigma(I_i) > \sigma(I_j)$.

**Definition 2.** Semi-dominance relation on instructions. For $I_i, I_j \in U$ and $Pf(I_i) = Pf(I_j)$, and not $I_i \prec I_j$ nor $I_i \prec E I_j$ nor $I_i \prec S E I_j$ nor $I_i \prec D I_j$, $I_i$ is said to semi-dominate $I_j$, denoted by $I_i \semdom I_j$, if $I_i, I_j$ satisfy one of the following conditions:

1. $S(I_i) \geq S(I_j)$ and $D(I_i) \geq D(I_j)$.
2. $S(I_j) = \emptyset$ and $L(I_i) \geq D(I_j) + 1$.

It can be proved that if $I_i \dom I_j$, $I_i$ is scheduled no later than $I_j$ is in an optimal schedule [17]. However, no analogous property can be proven for the semi-dominance relation. Nevertheless, if $I_i \semdom I_j$, $I_i$ is likely to be scheduled no later than $I_j$.

Thus in designing the ECG-algorithm, we adopt the following guidelines:

**G1:** If $I_i \dom I_j$, then $\alpha(I_i) > \alpha(I_j)$.

**G2:** If $I_i \semdom I_j$, then $\alpha(I_i) > \alpha(I_j)$.

The ECG-algorithm involves two major procedures: **leveling** and **labeling**. Let $L(I^*)$ denote the level of $I^*$, and $N(I^*)$ denote the decreasing sequence of integers formed by ordering the set $\{ \alpha(I') \mid I' \in S(I^*) \}$. The leveling procedure is then as follows:

**ECG-algorithm-leveling procedure**

/* input: instruction DAG */
/* output: leveled instruction DAG */
1. If \( I_i \) has no successors, then \( L(I_i) = D(I_i) + 1 \);
2. Else \( L(I_i) = \max \{L(I^*)\} + D(I_i) + 1, \forall I^* \in S(I_i) \).

**ECG-algorithm-labeling procedure**

/* input: leveled instruction DAG */
/* output: labeled instructions DAG */
1. \( lv = 1; i = 1 \);
2. for \( lv = 1 \) to \( \text{max-Level} \)
3. Let \( R \) be the set of instructions of level \( lv \);
4. while \( R \neq \emptyset \)
5. Let \( I^* \) be the instruction in \( R \) such that \( N(I^*) = \min \{N(I)\}, \forall I \in R \) (break tie at will);
6. \( \alpha(I^*) = i \);
7. \( R = R - \{I^*\} \);
8. \( i = i + 1 \);
9. endwhile;
10. endfor;

The labels of the instruction in Figure 1 are denoted by the subscripts. Figure 2 shows an ECG-schedule of the DAG in Figure 1.

![Figure 2. An ECG-schedule of the sample DAG.](image)

It can be proved that instructions labeled by the ECG-algorithm satisfy guidelines G1 and G2. Since the proof is not difficult, it is omitted here. Interested readers may refer to [17]. Let \( w_{\text{ECG}} \) denote the length of an ECG-schedule and \( w_{\text{opt}} \) the length of an optimal schedule. For a given \((k, M, \mathscr{P}, z)\), the worst-case performance of the ECG-algorithm is to find the value

\[
\max_{\forall(u, \prec, \mathscr{P}, z)} \left( \frac{w_{\text{ECG}}}{w_{\text{opt}}} \right).
\]

The significance of the heuristic algorithm is that its worst case bound is smaller than the worst-case bound of other known scheduling algorithms.
4. Previous work.

In this section, we briefly review relevant known results concerning the above problem. Some of these results concern UET task scheduling problems. We include them because the task scheduling problem and the instruction scheduling problem have some similarities. For scheduling UET tasks on \( m \) processors, the worst case of an arbitrary greedy algorithm is bound by

\[
\frac{w_{\text{greedy}}}{w_{\text{opt}}} \leq 2 - \frac{1}{m}.
\]

If the tasks are typed, there is a best possible bound, established by Jaffe [11]:

\[
\frac{w_{\text{greedy}}}{w_{\text{opt}}} \leq k + 1 - \frac{1}{\max \{m_i\}}
\]

when \( k \) is the number of typed processors.

Hu [12] proved that executing the tasks in a highest-level-first order results in an optimal schedule when precedence relations define an in-forest or out-forest. For an arbitrary DAG, Hu showed that the bound is

\[
\frac{w_{\text{Hu}}}{w_{\text{opt}}} \leq \begin{cases} 
\frac{4}{3} & \text{if } m = 2, \\
2 - \frac{1}{m} - 1 & \text{if } m > 2.
\end{cases}
\]

Coffman and Graham [13] proposed an algorithm (the CG-algorithm) which is optimal when \( m = 2 \). Lam and Sethi [14] explored the upper bound of the CG-algorithm. They established a tight worst-case bound:

\[
\frac{w_{\text{CG}}}{w_{\text{opt}}} \leq 2 - \frac{2}{m} \quad \text{for } m \geq 2.
\]

Bernstein and Gertner [15] showed that optimal scheduling of expressions on the machine \((1, M = (m_1), \mathcal{P}, z = 1)\) can be carried out by a slightly modified CG-algorithm. E. Lawler et al. [16] showed that the completion time of a schedule constructed using a highest-level-first greedy heuristic is arbitrarily close to \( 2 - 2/(m(z + 1) + 1) \) times optimal. However, their assumption of equal delay cycles is too restrictive for superscalar instructions. In this paper, we establish that the bound value for the ECG-algorithm is

\[
\frac{w_{\text{ECG}}}{w_{\text{opt}}} \leq k + 1 - \frac{2}{\max \{m_i\}(z + 1)}.
\]

Moreover, our model is very close to the actual superscalar instruction scheduling problem.

5. Upper bound analysis.

The general steps for finding the upper bound are the following:

1. Given a DAG and its ECG-schedule with length \( w_{\text{ECG}} \), find the minimal length \( w_{\text{opt}} \) required to schedule the DAG;
2. find the number of instructions and the number of idle cycles in the ECG-schedule;
3. Then \( w_{\text{ECG}}/w_{\text{opt}} \leq (\text{no. of ins.} + \text{no. of idle cycles})/(\text{no. of ins.}) \).

The key point usually lies in finding \( w_{\text{opt}} \).

To find \( w_{\text{opt}} \), we divided an ECG-schedule into segments, such that all the instructions in one segment must be completed before any instruction in the next segment can begin. Hence an optimal schedule corresponding to an ECG-schedule will be no shorter than one that arranges the instructions in each segment optimally. To identify segments, it would be easier first to identify the blocks which constitute segments. The concepts of a segment and block are borrowed from [14].

5.1. Block and segment partitioning.

Assume that if a processor does not issue a new instruction in time slot \( t_i \), then it issues a dummy instruction with label 0 in \( t_i \). Blocks \( B_1, \ldots, B_0 \), for some \( i \geq 0 \), are defined as follows:

**Block partitioning procedure.**
1. Let \( b_0 \) be the last finished instruction.
2. For \( i \geq 1 \), \( b_i \) is defined to be the most recent instruction issued by \( P_{t_i} \), such that \( \lambda(b_i) + D(b_i) = t_i \), \( \alpha(b_i) > \alpha(b_{i-1}) \) and there is no other instruction whose label is greater than \( b_{i-1} \) and whose scope contains \( t_i \).
3. Repeat step 2 until no more \( b_i \) can be found.
4. Block \( B_{i-1} \) is the set of all instructions \( I^* \) satisfying \( \lambda(b_i) + D(b_i) < \lambda(I^*) \leq \lambda(b_{i-1}) + D(b_{i-1}) \) and \( \alpha(I^*) \geq \alpha(b_{i-1}) \).

Suppose the above steps define \( b_i \) for \( 0 \leq i \leq q \) and \( b_{q+1} \) does not exist. Then \( B_q \) is the set of instructions \( I^* \) such that \( \lambda(I^*) \leq \lambda(b_q) + D(b_q) \) and \( \alpha(I^*) \geq \alpha(b_q) \).

The block boundaries and block instructions of the sample ECG-schedule are surrounded by bold lines in Figure 3. Instructions belonging to a block are called block instructions. Unless otherwise specified, instructions mentioned in the rest of this paper refer to block instructions.

After block partitioning, we may observe the following features:
1. There may be some instructions that fail to belong to any block. Such instructions are called extra instructions. For example, instruction \( I_9 \) in Figure 3 is an extra instruction.
2. The last time slot of \( B_i \) is in the scope of \( b_i \) only, the other slots are contained by at least two instructions in \( B_i \).
3. All the instructions in \( B_i \) depend on \( b_{i+1} \). For example, \( I_{20} \) and \( I_{10} \) precede all the instructions scheduled behind them.
Figure 3. An ECG-schedule and its blocks of the sample DAG.

It is not always true that all instructions in one block precede all instructions in the next block. For example, instruction $I_{11}$ in $B_1$ does not precede every instruction in $B_0$.

Next, blocks are used to form segments. Segments are formed by scanning blocks from left to right. If any instruction in a block on the left does not precede all the instructions in the next block on the right, then the two blocks are merged into one segment, or else a segment is terminated and a new segment begun. The important feature in forming segments is that when two blocks are to be merged, an extra instruction can always be found. The extra instruction has to be counted into the total number of instructions. For example, Figure 4 indicates the segments of the sample ECG-schedule. There are two segments in the figure. Segment 1 consists of block 2, since all the following instructions depend on every instruction in block 2. Segment 2 consists of block 1 and block 0, since $I_{11}$ does not precede each instruction in block 0. Since it is tedious to describe the procedure for forming segments and the proof that extra instructions can always be found, we shall not do so here. Interested readers may refer to [17].

5.2. Minimal length of segments.

After finding segments, we can obtain an optimal schedule of a set of instructions by scheduling each segment optimally. So instead of calculating the worst-case ratio of an entire schedule, we calculate the worst-case ratio of an optimal segment to an ECG-segment.

For block instructions, an ECG-schedule includes three kinds of time slots: full slots, partial slots, and delay slots.
- A full slot $t_f$ is a time slot in which all the processors of a certain type are issuing new instructions. Time slots (1, 4, 5, 9) are such slots.
- A delay slot $t_d$ is a time slot in which all processors are idle. Time slots (3, 7, 8, 10, 11) are delay slots.
- A partial slot $t_p$ is a time slot in which at least one instruction is issued and at least one processor of each type is idle. Time slots (2, 6) are partial slots.
This classification of time slots is intended to reveal dependencies between instructions. Since the maximal delay cycle is $z$ and the scheduling discipline is greedy, we have the following dependencies associated with partial and delay slots.

**Property 1.** In an ECG-schedule, all instructions scheduled after a delay slot $t_d$ depend on at least one instruction scheduled in $(t_d - z, t_d - 1)$.

**Property 2.** In an ECG-schedule, all instructions scheduled after a partial slot $t_p$ depend on at least one instruction scheduled in $(t_p - z, t_p)$.

**Lemma 1.** Let $W$ be a segment of an ECG-schedule. If $W$ contains $d$ delay slots and $p$ partial slots, then $w_{opt} \geq p + d + f'$, where $f' \geq 0$ and $p + f' \geq d/z$.

**Proof.** This lemma can be proved by finding a chain $h$ in $W$ and showing that $\text{Len}(h) \geq p + d + f'$. Consequently, $w_{opt} \geq \text{Len}(h) \geq p + d + f'$.

Suppose $h = \langle I_1, I_2, \ldots, I_x \rangle$, then $h$ can be constructed from the end of $W$ by chaining instructions to $h$ one at a time until no more instructions can be chained. The chain construction steps are as follows:

1. Let $I_x$ be the last completed instructions in $W$.
2. Suppose $i$ instructions have been chained for $i \geq 1$. $I_{(x-i)y}$ is selected in such a way that $I_{(x-i)y}$ is the rightmost (in terms of completion time) immediate predecessor of $I_{(x-(i+1)y)}$ (break tie at will).
3. Repeat step 2 until no more instructions can be chained.

If we take $W_2$ in Figure 4 as an example, the chain constructed by the steps above is $h = \langle I_{18}, I_{14}, I_{10}, I_8 \rangle$ (notice that $h$ is not unique in this case), which has a length of 8 cycles.

It can be proved that the scopes of the instructions in $h$ cover all $p$ partial and $d$ delay slots as well as any full slots. Thus $\text{Len}(h) \geq p + d + f'$, where $f' \geq 0$ denotes the number of full slots covered by the scopes of $h$. 

![Figure 4. Segments of the sample ECG-schedule.](image-url)
Next, we consider the minimal number of instructions in the chain. Since the maximal delay cycle is \( z \), there are no consecutive delay slots greater than \( z \) in \( W \). Thus for \( d \) delay slots, there must be at least \( \lceil d/z \rceil \) instructions in the chain and these instructions reside in partial or full slots, so \( p + f' \geq d/z \) (since \( p + f' \) is integer, the ceiling operation can be canceled).

![Figure 5. Packages of the ECG-schedule.](image)

However, this minimum length may be underestimated. To explore a greater minimum length of \( W \), we shall introduce the concept of packages. Packages are used to expose the detailed dependence relationships inside a segment. Packages are constructed by searching from right to left of \( W \) for partial or delay slots. Each time a partial or a delay slot is found, that slot and the next \( z \) slots to its left form a package. Let \( PK_i \) be used to denote the \( i \)th package. Figure 5 indicates the packages of the schedule in Figure 3. Since we start each package by finding a partial or delay slot, all \( p \) partial and \( d \) delay slots are covered by packages. From Property 1 and Property 2, each instruction scheduled after a package must depend on at least one instruction in that package. This fact encourages us to reexamine the minimal length of \( W \). We categorize packages into three types:

- \( A_1 \)-type packages are packages containing only full and delay slots.
- \( A_2 \)-type packages are packages containing one full slot, one partial slot with only one instruction scheduled in it, and any other full or delay slots.
- \( A_3 \)-type packages are the remaining cases, which include the following possibilities:
  1. one partial slot with one instruction and delay slots, or
  2. one partial slot with two instructions and any other slots, or
  3. two partial slots, each with one instruction, and any other slots.

For example, the types of the three packages in Figure 5 are as follows: \( PK_1 \) of \( W_1 \) is \( A_1 \)-type, since time slot 1 is a full slot, while \( PK_1 \) and \( PK_2 \) of \( W_2 \) are \( A_3 \)-type.

**Lemma 2.** Let \( W \) be a segment of an ECG-schedule with \( p \) partial and \( d \) delay slots.
If there are $a_1$ $A_1$-type packages and $a_2$ $A_2$-type packages in $W$, then $w_{opt} \geq (p + d + a_1 + \lceil a_2/2 \rceil)$.

**Proof.** From the proof of Lemma 1, we can find a chain $h$ whose scopes cover all $p$ partial and $d$ delay slots in $W$. But the length of $h$ equals $p + d$ only when its scopes cover just the $p$ partial and the $d$ delay slots. If the scopes of $h$ cover any full slots, these full slots have to be taken into account as contributing to the length of $h$.

First, we prove that $A_1$-type packages contribute to the minimal completion time of $W$. Suppose $PK_i$ is an $A_1$-type package containing only full and delay slots (by definition). From the dependency properties, every instruction scheduled after $PK_i$ must depend on at least one instruction in $PK_i$, hence it must depend on an instruction scheduled in a full slot. So if there are $a_1 A_1$-type packages, the scopes of $h$ cover $a_1$ full slots. The minimal completion time of $W$ is thus $p + d + a_1$ cycles.

Next, we show that $A_2$-type packages also contribute to the minimal completion time of $W$, but the minimal contribution is $\lceil a_2/2 \rceil$. Suppose without loss of generality that there are only one full slot and one one-instruction partial slot in each $A_2$-type package. As indicated in Figure 6, let $t_{f1}$ denote the full slot, $t_{p1}$ the partial slot, $I_{f1}$ the set of the instructions scheduled in $t_{f1}$, and $I_{p1}$ the only instruction scheduled in $t_{p1}$.

**Case 1.** $t_{f1} > t_{p1}$.

Figure 6 shows such a case. Suppose $t_{f1}$ is not in the scopes of $h$ but $t_{p1}$ is. Since we start each package by finding a partial or a delay slot, there must be a delay slot after $t_{f1}$. If the scopes of $h$ terminated before $t_{f1}$, the successors of $I_{p1}$ could be scheduled in the delay slot — a contradiction. If $I_{p1}$ is the last instruction on $h$, its completion time should cover $t_{f1}$, otherwise $I_{p1}$ could not be chosen when constructing $h$. So the scopes of $h$ must contain $t_{f1}$.

**Case 2.** $t_{f1} < t_{p1}$.
Figure 7 indicates this situation. We prove by induction that if there are \( a_2 \) packages of this kind, they contribute \( \left\lfloor a_2/2 \right\rfloor \) to the minimal length of \( W \). Let \( m \) denote any \( m_i \) for \( 1 \leq i \leq k \). We claim that when \( a_2 \) is odd, there are \((m + 1) \times [p + d + (a_2 - 1)/2]\)-cycle chains or a \([p + d + (a_2 + 1)/2]\)-cycle chain in \( W \). In both situations, \( W \) needs at least \( p + d + (a_2 + 1)/2 \) cycles to be completed. When \( a_2 \) is even, there must be a \([p + d + a_2/2]\)-cycle chain in \( W \).

First, let \( a_2 = 1 \). Taking Figure 7 as an example, we prove that there are \((m + 1) \times (p + d)\)-cycle chains or a \((p + d + 1)\)-cycle chain in \( W \). If \( t_f_1 \) is in the scope of \( h \), the scope of \( h \) contain a full slot and a \((p + d + 1)\)-cycle chain is found, so the proof is completed. Suppose \( t_f_1 \) is not in the scope of \( h \). Since the scope of \( h \) terminates before \( t_f_1 \), the only reason that prevents \( I_p \) from being scheduled in \( t_f_1 \) must be \( (I_f_1) > (I_p) \), \( \forall I_f_1 \). From the scheduling discipline, \( L(I_f_1) \geq L(I_p) \) must hold. Now consider the length of the chains going through \( I_f_1 \). The chains cover all the partial and delay slots before \( t_f_1 \). The level of \( I_f_1 \) is not less than \( I_p \), so in turn, it is not less than the summation of the numbers of partial and delay slots after \( t_f_1 \). Therefore, the lengths of the chains going through \( I_f_1 \) are at least \((p + d)\)-cycles long. So there are \( m + 1 \times (p + d)\)-cycle chains in \( W \). \( W \) needs at least \( p + d + 1 \) cycles to be completed.

Second, let \( a_2 = 2 \). We prove that there is a \((p + d + 1)\)-cycle chain in \( W \). Figure 8 illustrates this situation. \( PK_1 \) and \( PK_2 \) in Figure 8 are two \( A_2\)-type packages. We must suppose \( t_f_1 \) and \( t_f_2 \) are not within the scope of \( h \), otherwise the proof is completed.

Since \( PK_2 \) does not cause a block split, not all instructions after \( t_p_2 \) depend on \( I_p_2 \) only. In other words, \( I_f_2 < I_p_1 \) or \( I_f_2 < I_f_1 \) must hold. If \( I_f_2 < I_p_1 \), then the length of the chain containing \( I_f_2 \) and \( I_p_1 \) is \( p + d + 1 \), because its scope contains all the \( p \) partial and \( d \) delay slots and the one full slot \( (t_f_2) \). If \( I_f_2 < I_f_1 \), the scope of the chain containing \( I_f_1 \) and \( I_f_2 \) cover all the partial and delay slots before \( t_f_1 \) and the one full slot \( (t_f_2) \). So the length of the chain is also \( p + d + 1 \).

For \( a_2 > 2 \), similar arguments can be applied, so this part of the proof is omitted. A complete proof can be found in [17].
Finally, we compare the contribution of the $A_2$-type packages in Case 1 and Case 2. We see that the contribution is $\lceil a_2/2 \rceil$ in Case 2 versus $a_2$ in Case 1. So we conclude that when there are $a_2$ $A_2$-type packages, the minimal length of $W$ is $p + d + \lceil a_2/2 \rceil$. The contributions of $A_1$-type and $A_2$-type packages to the minimal length of $W$ are independent. Thus their contributions can be added.

**Lemma 3.** Given segment $W$ of an ECG-schedule, if there are $a_2$ $A_2$-type and $a_3$ $A_3$-type packages in $W$, the number of extra instructions plus instructions in partial slots is not less than $a_2 + 2a_3 - 1$.

**Proof.** We first prove this lemma when $W$ contains only one block. Then we show that it is also true when $W$ contains more than one block.

Assume that $W$ contains only one block, and consider the number of instructions in $A_3$-type packages. Since $W$ has only one block, there can be only one $A_3$-type package ($P K_1$) that contains only one instruction. The other $A_3$-type packages must contain at least two instructions in partial slots. For $A_2$-type packages, by definition, each $A_2$-type package contains one instruction in a partial slot. Therefore the total number of instructions scheduled in partial slots of $W$ is at least $a_2 + 2a_3 - 1$ ($A_1$-type packages have no partial slots).

If $W$ contains $B > 1$ blocks, then there could be $B$ $A_3$-type packages which contain one instruction only (the rightmost package of each block). But there are $B - 1$ extra instructions to complement these one-instruction packages. So the number of extra instructions plus the instructions in partial slots is still $a_2 + 2a_3 - 1$.

**Theorem 1.** Given $W$, a segment of an ECG-schedule $G$ on a $(k, M, \mathcal{P}, z)$, let $w_{ECG}$ be the length of $W$ and $w_{opt}$ the length of an optimal schedule of $W$. Then $w_{ECG}/w_{opt} \leq k + 1 - 2/(\max\{m_i\}(z + 1))$. 

Figure 8. Two $A_2$-type packages with partial slots behind full slots.
PROOF. We see that the worst-case ratio of $w_{ECG}/w_{opt}$ exists in optimal schedules that contain no idle cycles. If an optimal schedule contains idle cycles, then these can be eliminated by inserting some independent instructions into the original DAG to fill up the idle cycles without affecting the length. This does not mean that the worst-case ratio always exists in an optimal schedule without idle cycles. It means that the search for the worst-case ratio can be narrowed down to those optimal schedules which have no idle cycle. By the above argument, we conclude that the first time slot of $W$ must be a full slot; otherwise the optimal schedule will contain idle cycles.

Let the number of partial slots in $W$ be $p$, the number of delay slots be $d$ and $\Mathcal{M} = \sum_{i=1}^{k} m_i$. Let there be $a_1 A_1$-type, $a_2 A_2$-type and $a_3 A_3$-type packages in $W$ and let $q = a_1 + a_2 + a_3$. Let $U_i$ denote the set of type $i$ instructions and $V_i$ the set of type $i$ instructions scheduled in partial slots.

CASE 1. $PK_q$ (the leftmost package) contains the first time slot.

SUBCASE 1. $PK_q$ is an $A_1$- or $A_2$-type package.

In this subcase, the number of partial and delay slots in $PK_q$ must be no greater than $(z + 1)/2$, since otherwise the optimal schedule of $W$ will contain idle cycles. Assume $PK_q$ have $d'$ delay and $p'$ partial slots if $PK_q$ is $A_2$-type, or $p' = 0$ if $PK_q$ is $A_3$-type. All instructions scheduled after $PK_q$ are preceded by a chain at least $d' + p'$ long. Thus only the instructions in $PK_q$ can be scheduled in the first $d' + p'$ slots in the optimal schedule. Notice that $PK_q$ has at most one instruction in a partial slot. If $d' + p' > (z + 1)/2$, the number of instructions in $PK_q$ is no more than $\Mathcal{M}(z - 1)/2 + 1$, which is obviously not enough to fill the first $d' + p'$ slots when $\Mathcal{M} > 1$.

According to Lemma 2, the minimal length of $W$ is $p + d + a_1 + \lceil a_2/2 \rceil$. Consider the time-space product

$$\Mathcal{M} w_{ECG} \leq \sum_{i=1}^{k} (\Mathcal{M} - m_i)(U_i - V_i)/m_i + \Mathcal{M} p - \sum_{i=1}^{k} V_i + \Mathcal{M} d + \Mathcal{M} w_{opt}$$

$$\leq \Mathcal{M}(k - 1)w_{opt} - \Mathcal{M} \sum V_i/m_x + \sum V_i$$

$$+ \Mathcal{M}(p + d + a_1 + \lceil a_2/2 \rceil - a_1 - \lceil a_2/2 \rceil - \sum V_i + \Mathcal{M} w_{opt}$$

$$w_{ECG}/w_{opt} \leq k + (p + d + a_1 + \lceil a_2/2 \rceil - a_1 - \lceil a_2/2 \rceil - \sum V_i/m_x)/w_{opt}.$$ 

Since $p \geq \sum V_i/m_x$, the numerator of the rightmost term on the right-hand side of the last equation is greater than zero, and $w_{opt} \geq (p + d + a_1 + \lceil a_2/2 \rceil)$. Thus we have

$$w_{ECG}/w_{opt} \leq (k + 1) - (a_1 + \lceil a_2/2 \rceil + \sum V_i/m_x)/(p + d + a_1 + \lceil a_2/2 \rceil)$$

since $(p + d + a_1 + \lceil a_2/2 \rceil) \leq (z(a_1 + a_2 - 1) + (z + 1)/2 + (z + 1)a_3 + a_1 + \lceil a_2/2 \rceil)$. (1)

$$w_{ECG}/w_{opt} \leq (k + 1) - \frac{(a_1 + \lceil a_2/2 \rceil + (a_2 + 2a_3 - 1)/m_x)}{z(a_1 + a_2 - 1) + (z + 1)/2 + (z + 1)a_3 + a_1 + \lceil a_2/2 \rceil}.$$
Figure 9. A worst-case DAG.

SUBCASE 2. \( PK_q \) is an \( A_3 \) type package. Using similar techniques we find

\[
(2) \quad \frac{w_{ECG}}{w_{opt}} \leq (k + 1) - \frac{(a_1 + \lceil a_2/2 \rceil + 1 + (a_2 + 2a_3 - 1)/m_x)}{z(a_1 + a_2) + (z + 1)a_3 + a_1 + \lceil a_2/2 \rceil + 1}.
\]

CASE 2. \( PK_q \) does not contain the first slot. Again we use a corresponding procedure; the result turns out to be the same as in (2).

To find the upper bound of \( w_{ECG}/w_{opt} \), we must maximize the expressions 1 and 2, that is, minimize the minus terms in them. Since the variables in these terms are all integers we have to use slightly unorthodox methods. In this way we find that these two minus terms cannot be smaller than \( 2/(z + 1)m_x \). Here \( a_1, a_2 \) and \( a_3 \) are variables while \( z \) and \( m_x \) are regarded as constants.

For expression 1:

\[
(a_1 + \lceil a_2/2 \rceil + (a_2 + 2a_3 - 1)/m_x)(z(a_1 + a_2 - 1))
+ (z + 1)/2 + (z + 1)a_3 + a_1 + \lceil a_2/2 \rceil - 2/(z + 1)m_x
= (a_1 + \lceil a_2 \rceil + (a_2 + 2a_3 - 1)/m_x)(z + 1)m_x -
2(z(a_1 + a_2 - 1) + (z + 1)/2 + (z + 1)a_3 + a_1 + \lceil a_2/2 \rceil)
= a_1(z + 1)(m_x - 2) + \lceil a_2/2 \rceil(m_x - 2) + zm_x[\lceil a_2/2 \rceil - za_2 + a_2 - 2]
\geq 0, \text{ for } m_x \geq 2.
\]
A similar approach can be applied to expression 2 to obtain the same result. So we conclude that $w_{ECG}/w_{opt} \geq k + 1 - 2/\max \{m_i\}(z + 1)$.

From the nature of segments, an optimal schedule can be found by scheduling each segment optimally. So the result of Theorem 1 can be applied to a whole schedule.

**Theorem 2.** Let $G$ be an ECG-schedule of a given $(U, \prec, Pf, D)$ on a given $(k, M, \mathcal{P}, z)$. Let $w_{ECG}$ be the length of $G$ and $w_{opt}$ be the length of an optimal schedule of the same $(U, \prec, Pf, D)$. Then $w_{ECG}/w_{opt} \leq k + 1 - 2/\max \{m_i\}(z + 1)$.

To show that the bound is the best possible, we give an example for $k = 2$, $z = 2$, and $(m_1 = 1, m_2 = 4)$. Consider the DAG in Figure 9. The DAG pattern contains $6n$ type 1 instructions (denoted by $a_i$) and $6n$ type 2 instructions (denoted by $J_i$). The type 1 instructions have no delay cycle but all type 2 instructions have two delay cycles. The index of each instruction is a valid ECG-algorithm label. The ECG-schedule of the DAG is shown in Figure 10, and an optimal schedule is shown in Figure 11, in which the tail of the first pattern is overlapped with the next pattern. If the DAG pattern is duplicated $x$ times, the length of the ECG-schedule is $x(6n + 11n)$, and the length of the optimal schedule is $x(6n + 1) + 6n$. The ratio is thus

$$\frac{w_{ECG}}{w_{opt}} = \frac{x(17n)}{x(6n + 1) + 6n} \approx \frac{17}{6}$$

when $x$ and $n$ are large. This value is in agreement with Theorem 2.
6. Conclusion.

A heuristic instruction scheduling algorithm called the ECG-algorithm is presented, and its worst-case performance is explored. A new technique called package partitioning is developed to expose the detailed dependence relationships between instructions. A bound of at most \((k + 1 - 2/\max \{m_i\}(z + 1))\) times the length of an optimal schedule is obtained. The bound is smaller than that of the highest-level first algorithm.

REFERENCES