Static Workload Balance Scheduling; Continuous Case

Sabin Tabirca‡, T. Tabirca†, Len Freeman†, Laurence Tianruo Yang§

†Department of Computer Science, University of Manchester
Oxford Road, Manchester, M13 9NG, UK
‡University College Cork, Department of Computer Science
College Road, Cork, Ireland
§Department of Computer Science, St. Francis Xavier University
P.O.Box 5000, Antigonish, NS, B2G 2W5, Canada

Abstract

This article studies a static scheduling method based on workload balancing in the continuous case. An equation is presented for the case when the workload, as continuous function, is equally distributed onto processors based on integrals. A sufficient condition is also established for the fully covering property. Finally, some computational results are given to prove that the continuous case is better than the discrete case.

1 Introduction

Parallel programming has been used intensely in order to solve problems with a large number of computations or large volumes of data. These problems naturally arise in real world applications (e.g. Weather Prediction) or theoretical applications (e.g. Differential Equations). Loops represent an important source of parallelism and occur in at most all the scientific applications. Many algorithms dealing with loop scheduling have been proposed so far.

Every loop scheduling method provides a mapping of the loop iterations to a number of processors. There are two important classes of loop scheduling. The first class contains static scheduling methods that finds this mapping at compile-time. These can also be grouped in block methods where successive iterations are mapped to a processor and cyclic methods where a processor receive loop iterations in a cyclic manner. A well-known static block scheduling is the uniform scheduling for which all the chunks have almost the same dimension.

The second class contains dynamic scheduling methods which decide the mapping of the loop iterations at run-time.

These methods use a set of queue structures and a strategy for taking loop iterations from them. The guided self-scheduling algorithms (Polychronopoulos et al. [7]) and some of theirs variants (Eager and Zahojan [3], Hummel et al. [4], Lucco [6]) have one single queue. The number of chunks is considerably greater than the number of processors and decreases dynamically while the chunks are taken from the queue and assigned to processors. Potential loss of performance may be caused by overheads such as loss of data locality, inefficient unrolling and pipelining etc. The affinity self-scheduling algorithms (Yang et al. [9]) avoid a part of these overheads by considering one single queue per processor.

A recent scheduling method named Feedback Guided Dynamic Loop Scheduling was proposed by Bull [1] to solve the scheduling of a sequence of similar parallel loops. The method divides the loop iterations into \( p \) chunks during the run-time so that each processor receives similar workload.

Based on this approach Tabirca et al. [8] proposed a static block scheduling method where the workloads are approximately equally distributed onto processors. A major inconvenience of this method is that the upper bounds of the scheduling partition do not have simple equations. This article addresses this issue and shows how to obtain simpler bounds if the continuous case is used.

2 Static Loop Scheduling Based on Workload Balance: Discrete Case

Tabirca et al. [8] proposed a static scheduling based on a balanced distribution of the workloads onto processors. The main results concerning this method are outlined in the following. Consider that there are \( p \) processors denoted in the following by \( P_1, P_2, \ldots, P_p \) and the single parallel loop.

We also assume that the workload of the routine...
do parallel i=1,n
    call loop_body(i)
end do

Figure 1. Single parallel loop

loop_body(i) can be evaluated and is given by the sequence $w_1, ..., w_n$, where $w_i$ represents either the number of the routine operations or its running time (presume that $w(0) = 0$). The static scheduling proposed in [8] is block where $l_j$ and $h_j$ are the lower and upper bounds of the iteration chunk assigned to Processor $j$, $j = 1, 2, ..., p$. The definition of these bounds considers that the workload is equally balanced onto the processors as follows:

$$\sum_{i=l_j}^{h_j} w_i \leq \frac{1}{p} \sum_{i=1}^{n} w_i := \overline{W}, \quad \forall j = 1, 2, ..., p. \quad (1)$$

Note that $\overline{W} = \frac{1}{p} \sum_{i=1}^{n} w_i$ represents the average workload per processors. This condition requires that each processor will receive a workload approximately equal with the average workload. The approximation from Equation (1) is evaluated by

$$h_j = h_j \iff \sum_{i=l_j}^{h_j} w_i \leq \overline{W} \leq \sum_{i=l_j}^{h_j} w_i, \quad (2)$$

These bounds also satisfy the following conditions

$$l_1 = 1, h_p = n, l_{j+1} = h_j + 1, j = 1, ..., p. \quad (3)$$

An important situation studied in [8] is when Equation (2) generates $p$ distinct upper bounds $h_1, h_2, ..., h_p$. In this case the loops are assigned onto all the processors or equivalently the processors are fully covered by the loops (situation named "Fully Covering"). Tabirca et al. [8] proved that if the workload satisfies

$$w_i \leq \overline{W} := \frac{1}{p} \sum_{k=1}^{n} w_k, \quad \forall i = 1, 2, ..., n, \quad (4)$$

the fully covering property holds.

Another important result proposed in [8] is an equation for the upper bounds $h_j$. If the partial sums of the workloads are $f(i) = \sum_{i=1}^{i} w_i, i = 1, 2, ..., n$ and $f_l$ is defined by $f_l(x) = n \iff f(n) \leq x < f(n + 1)$ then the upper bounds are given by

$$h_j = f_l(\overline{W} + f(h_j - 1)), j = 1, 2, ..., p, \quad (5)$$

where $h_0 = 0$. Equation (5) generates the upper bounds in $O(p)$ providing that the functions $f$ and $f_l$ are known. The following cases illustrate the equation of the upper bounds for a few particular workloads.

Example 1 The workload is a constant function $w_i = C$, $i = 1, 2, ..., n$. The fully covering property holds when $p \leq n$ since $w_i = C \leq \frac{1}{p} \sum_{i=1}^{n} w_i = \frac{nC}{p}$. Since $f(i) = \sum_{i=1}^{i} w_i = i \cdot C$ and $f_l(x) = \left[ \frac{x}{C} \right]$, the upper bounds are given by

$$h_j = \left[ \frac{h_{j-1} + n}{p} \right], j = 1, 2, ..., p. \quad (6)$$

Example 2 The workload is given by $w_i = i$, $i = 1, 2, ..., n$. The fully covering property holds when $p \leq \frac{n+1}{2}$ since

$$w_i = i \leq n \leq \sum_{i=1}^{n} w_i = \frac{n \cdot (n + 1)}{2} \cdot p.$$

We have $f(i) = \sum_{i=1}^{i} w_i = \frac{i(i+1)}{2}$ and $f_l(x) = \left[ \frac{-1 + \sqrt{1 + 4x}}{2} \right]$ (see [8]). Therefore, the upper bounds are given by

$$h_j = \left[ \frac{\sqrt{n(n+1)} + h_{j-1}(h_{j-1}+1) - \frac{1}{4} - \frac{1}{4}}{p} \right]. \quad (7)$$

As the above examples show, Equation (5) does not give simple expression for the bounds. Despite that, Tabirca et al. [8] proposed equations for some of the polynomial cases encountered in real problems. However, if the workloads $w_i, i = 1, 2, ..., n$ have complicated equations, it is quite impossible to find the functions $f$ and $f_l$ therefore, the method cannot be applied.

3 Static Loop Scheduling Based on Workload Balance: Continuous Case

In this section we will present an approach to overcome the main inconvenience of Equation (5). This is based on a similar approach that Bull et al. [2] applied for Feedback Guided Dynamic Loop Scheduling. For an equation similar with (1) they considered the sums as integrals by treating the loop index as a continuous variable. This approach is used in the following to achieve simpler bounds for the scheduling.

Let us suppose that the workloads $w_1, w_2, ..., w_n$ of the parallel loop are extended to a continuous function $w : [0, n] \rightarrow [0, \infty)$ such that $w(0) = 0$ and $w(n) = w_i, i = 1, 2, ..., n$. In this case the total workload of parallel computation is $\int_0^n w(i)di$ so that the average workload per processor is $\overline{W} = \frac{1}{p} \int_0^n w(i)di$. In this continuous case there exist the real bounds $0 = x_0 < x_1 < ... < x_p = n$ such that

$$\int_{x_{j-1}}^{x_j} w(i)di = \frac{1}{p} \cdot \int_0^n w(i)di := \overline{W}, \quad \forall j = 1, ..., p. \quad (8)$$
Certainly, Equation (8) represents the extension of Equation (1) from the discrete case to the continuous case. The major advantage of the former is that the real bounds can be exactly calculated. Based on them the lower and upper bounds of the static scheduling can be found by

\[ h_j = [x_j], \quad l_{j+1} = [x_j] + 1, \quad j = 0, 1, \ldots, p - 1. \]  

(9)

Since, the function \( w \) is continuous we find that \( W : [0, n] \to [0, W(n)] \) defined by \( W(x) = \int_0^x w(i) di \) is an invertible function, therefore let \( W^{-1} : [0, W(n)] \to [0, n] \) be its inverse.

**Theorem 3** The bounds \( \{x_j, \ j = 0, \ldots, p\} \) are given by

\[ x_j = W^{-1} \left( \frac{j}{p} \cdot W(n) \right), \ j = 0, 1, \ldots, p, \]  

(10)

so that the upper bounds of the scheduling are

\[ h_j = \left[ W^{-1} \left( \frac{j}{p} \cdot W(n) \right) \right], \ j = 1, \ldots, p. \]  

(11)

**Proof.** Equation (8) is simply rewritten as follows

\[ \int_0^{x_j} w(i) di = \frac{j}{p} \cdot \int_0^n w(i) di, \]

which gives directly that

\[ W(x_j) = \frac{j}{p} \cdot W(n). \]

Since \( W \) is invertible we find \( x_j = W^{-1} \left( \frac{j}{p} \cdot W(n) \right) \). \( \triangle \)

Theorem 3 gives a simple equation for the upper bounds \( \{h_j, \ j = 1, \ldots, p\} \). More importantly, we can find formulas for the functions \( W \) and \( W^{-1} \) in most of the practical cases. One case that is very important considers a polynomial workload given by \( w(i) = i^p \).

**Corollary 4** If the workload is \( w : [0, n] \to R \) defined by \( w(i) = i^p \) then the upper bounds are

\[ h_j = \left[ \left( \frac{j}{p} \right)^{\frac{1}{p+1}} \cdot n \right], \ j = 1, 2, \ldots, p. \]

(12)

The proof comes directly from Equation (11) based on \( W(x) = x^{a+1} \) and \( W^{-1}(y) = ((a + 1) \cdot y)^{\frac{1}{a+1}} \).

### 3.1 The Fully Covering Property

The real bounds \( \{x_j, \ j = 0, 1, \ldots, p\} \) given by Equation (8) are used in fact to obtain the integer bounds \( \{h_j, \ j = 1, \ldots, p\} \) based on \( h_j = [x_j], \ j = 1, 2, \ldots, p \). Processor \( P_j \) executes \text{loop body}(i) \text{ for } i = l_j, \ldots, h_j. \) An extreme, and unwanted, case is when two consecutive processors \( P_j \) and \( P_{j+1} \) receive the same block of loops. This situation occurs when

\[ [x_j] = [x_{j+1}]. \]  

(13)

Provided that Equation (13) does not hold then each processor receives a different block of loops. In this case the set of processors is fully covered with workload.

In this section we propose a condition under which the fully covering property holds. Consider that the workload \( w : [0, n] \to [0, \infty) \) satisfies

\[ w(x) < \frac{1}{p} \cdot \int_0^n w(i) di, \ \forall x \in [0, n]. \]  

(14)

Equation (14) expresses the simple fact that there is no loop iteration for which the workload is bigger that the average \( \overline{W} = \frac{1}{p} \cdot \int_0^n w(i) di \).

**Theorem 5** If \( w : [0, n] \to [0, \infty) \) satisfies Equation (14), then

\[ x_{j+1} \geq x_j + 1, \ j = 0, 1, \ldots, p - 1, \]  

(15)

**Proof.** Assume that \( \exists j = [0, p-1] \) such that \( x_{j+1} < x_j + 1 \). Equation (8) gives that

\[ \int_{x_j}^{x_{j+1}} w(i) di = \overline{W}. \]

The mean value theorem gives that \( \exists c \in [x_j, x_{j+1}] \) such that

\[ w(c) = \frac{1}{x_{j+1} - x_j} \int_{x_j}^{x_{j+1}} w(i) di. \]

Since \( x_{j+1} - x_j < 1 \) we find that

\[ w(c) = \frac{1}{x_{j+1} - x_j} \int_{x_j}^{x_{j+1}} w(i) di > \int_{x_j}^{x_{j+1}} w(i) di = \overline{W}, \]

therefore the workload \( w(x) \) satisfies \( \exists c \in [x_j, x_{j+1}], \ w(c) < \overline{W}. \) This contradicts Equation (14) so we can conclude that \( x_{j+1} \geq x_j + 1 \) holds for all \( j = 0, 1, \ldots, p - 1. \) \( \triangle \)

Since \( x_{j+1} \geq x_j + 1 \), we find that \( [x_{j+1}] \geq [x_j + 1] = [x_j] + 1. \) Therefore, \( h_{j+1} = [x_{j+1}^f] = [x_j^f] + h_j \) holds, which implies the fully covering property.

**Corollary 6** For the workload is \( w : [0, n] \to R \) defined by \( w(i) = i^p \) if \( p < \frac{n}{a+1} \) then the covering property holds.

This corollary is a direct application of Theorem 5.
4 Comparisons Between the Discrete and Continuous Cases

This section presents some computational results for comparing the discrete and continuous cases. Recall that the static block scheduling based on workload balance distributes equally the workloads onto processors. The discrete case finds the bounds by using Equation (5). This equation has two important drawbacks that make it difficult to apply. The first one is that \( h_j \) depends on \( h_{j-1} \) therefore the bounds of the methods are found in \( O(p) \) or even \( O(\log p) \) using a special parallel computation [5]. But the most important inconvenience is that the functions \( f \) and \( f[j] \) cannot be computed. Moreover, the equations of them are quite complicated even for simple workload. Secondly, the continuous case proposes Equation (11) to compute the upper bounds. This is far simpler than Equation (5) and gives a parallel computation of the bounds in \( O(1) \). More importantly, the function \( W \) and \( W^{-1} \) are not quite difficult to be found even for complicated workloads. Therefore, the continuous case offers a simpler method to find the bounds.

It remains to see which of these two scheduling methods offers better computational times. Firstly, we do a synthetic comparison for the simple workloads \( w : [0, n] \rightarrow [0, \infty) \), \( w(i) = i \) where \( n = 1000 \). It is clear that the bounds \( \{ (l_j, h_j), j = 1, 2, ..., p \} \) generate a block scheduling with the execution time \( T = \max_{j=1}^{p} \sum_{i=l_j}^{h_j} i \). For our comparison we consider the following block scheduling:

- **Uniform**: The bounds of the uniform block scheduling are
  \[
  h_j^u = \left[ \frac{j \cdot n}{p} \right], \quad j = 1, 2, ..., p \tag{16}
  \]
  and generate the execution time \( T^u = \max_{j=1}^{p} \sum_{i=l_j^u}^{h_j^u} i \).

- **Workload balance (discrete case)**: The bounds are
  \[
  h_j^d = \left[ \sqrt{\frac{n(n+1)}{p} + h_{j-1}(h_{j-1}+1) - \frac{1}{4}} - 1 \right],
  \tag{17}
  \]
  and generate the execution times \( T^d = \max_{j=1}^{p} \sum_{i=l_j^d}^{h_j^d} i \).

- **Workload balance (continuous case)**: The bounds are
  \[
  h_j^c = \left[ n \cdot \sqrt{\frac{j}{p}} \right], \quad j = 1, 2, ..., p \tag{18}
  \]
  and the execution time is \( T^c = \max_{j=1}^{p} \sum_{i=l_j^c}^{h_j^c} i \).

The values of the synthetic times \( T^u \), \( T^d \) and \( T^c \) are presented in Table 1.

<table>
<thead>
<tr>
<th>Method</th>
<th>( p = 1 )</th>
<th>( p = 2 )</th>
<th>( p = 4 )</th>
<th>( p = 8 )</th>
<th>( p = 16 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T^u )</td>
<td>500500</td>
<td>375250</td>
<td>218875</td>
<td>117250</td>
<td>61047</td>
</tr>
<tr>
<td>( T^d )</td>
<td>500500</td>
<td>250929</td>
<td>125955</td>
<td>65722</td>
<td>36334</td>
</tr>
<tr>
<td>( T^c )</td>
<td>500500</td>
<td>250278</td>
<td>125250</td>
<td>62966</td>
<td>31590</td>
</tr>
</tbody>
</table>

**Table 1. Synthetic Times \( T^u \), \( T^d \) and \( T^c \)**

Figure 2. Variation of the synthetic times

The real problem analyzed is the product between an upper diagonal matrix \( a \in M_n(R) \) and a vector \( x \in R^n \). The product \( y = A \cdot x \in R^n \) is given by the equations

\[
  y(i) = \sum_{k=1}^{i} a(i,k) \cdot x(k), \quad i = 1, 2, ..., n. \quad (19)
\]

Equation (19) gives the following parallel computation.

\[
  \text{do parallel } i=1,n \\
  \text{do } k=1,i \\
  \quad y(i)=y(i)+a(i,k)*x(k) \\
  \text{end do} \\
  \text{end do}
\]

The workloads of this computation can be evaluated by \( w_i = O(i), \quad i = 1, 2, ..., n \), thus the upper bounds from Equations (16,17,18) can be used. All three scheduling methods were executed on a SGI Origin 2000 parallel machine with 16 processors. The running times for \( n = 1000 \) are presented in Table 2.

<table>
<thead>
<tr>
<th>Method</th>
<th>( p = 1 )</th>
<th>( p = 2 )</th>
<th>( p = 4 )</th>
<th>( p = 8 )</th>
<th>( p = 16 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T^u )</td>
<td>2.028</td>
<td>2.122</td>
<td>1.335</td>
<td>0.771</td>
<td>0.534</td>
</tr>
<tr>
<td>( T^d )</td>
<td>2.045</td>
<td>1.377</td>
<td>0.760</td>
<td>0.472</td>
<td>0.321</td>
</tr>
<tr>
<td>( T^c )</td>
<td>2.032</td>
<td>1.305</td>
<td>0.734</td>
<td>0.445</td>
<td>0.312</td>
</tr>
</tbody>
</table>

**Table 2. Execution times for the matrix-vector product**
Figure 3. Variation of the running times

We can draw two important remarks from Figures 2, 3. Firstly, the times of the uniform scheduling reflect a poor load balance since the last chunk contains all the biggest workload. Secondly, the workload balance scheduling offers quite similar times. However, the workload balance scheduling in the continuous case is better than in the discrete case generating better times.

5 Conclusions

This article has presented a version of the workload balance scheduling method in the continuous case. The continuous approach has considered the parallel loop index $i$ as a continuous variable and the workloads as a continuous positive function $w : [0, n] \rightarrow [0, \infty)$. The equation $h_j = W^{-1}(\frac{1}{p} \cdot W(n))$ has been proposed for the upper bounds of the method. A sufficient condition has been established for the fully covering property of the scheduling.

The article has compared the workload balance scheduling in the continuous and discrete cases when the workloads are given by $w : [0, n] \rightarrow [0, \infty)$, $w(i) = i$. The workload balance scheduling in the continuous case has provided better results both in synthetic and parallel computation. Unfortunately, a theoretical result for the comparison of $T_d$ and $T_c$ has not been found yet.

References


