An $O(logp)$ Algorithm for the Discrete Feedback Guided Dynamic Loop Scheduling

Tatiana Tabirca* Sabin Tabirca †
University College Cork, Department of Computer Science, BCRI
College Road, Cork, Ireland.
s.tabirca@cs.ucc.ie

Laurence Tianruo Yang
St. Francis Xavier University, Department of Computer Science
P.O.Box 5000, Antigonish, B2G 2W5, Canada.
lyang@stfx.ca

Abstract

In this paper we investigate a new algorithm for the Feedback-Guided Dynamic Loop Scheduling (FGDLS) method in the discrete case. The method uses a feedback-guided mechanism to schedule a parallel loop within a sequential outer loop. The execution times and the scheduling bounds for the current outer iteration are used to find the scheduling bounds of the next outer iteration. An $O(p + \log p)$ algorithm has been proposed for the discrete case where it was proved to achieve optimal bounds in only a few iterations. This articles introduces an $O(\log p)$ algorithm for the discrete case and presents some properties of it.

1 Introduction

Load imbalance is observed to be a significant overhead in many parallel implementations. Further, since loops are the dominant source of parallelism for many applications, a variety of algorithms that aim to schedule loop iterations to processors of a shared-memory machine in an almost optimal way (so-called loop scheduling algorithms) have been designed.

An important class of scheduling algorithms is based on *Guided Self-Scheduling* (Polychronopoulos and Kuck [9]) or some variant of *Guided Self-Scheduling* (see for example Eager and Zahorjan [4], Hummel et al. [5], Lucco [7], Tzen and Ni [13]). These algorithms split the loop iterations into a large number of chunks which are assigned to processors under the control of a central queue control. One of the motivations for this approach is the assumption that each execution of a loop is independent of any previous executions of the same loop, and therefore has to be rescheduled 'from scratch'. One effect of this approach is a potential loss of performance, caused by overheads such as additional synchronization, loss of data locality, and reductions in the efficiency of loop unrolling and pipelining. An attempt to improve this loss of performance has led to the class of *Affinity Scheduling* algorithms (Markatos and LeBlanc [8], see also Subramanian and Eager [10], and Yan et al. [14] for variants of *Affinity Scheduling*). Rather than maintaining a single work queue these algorithms are based on per-processor work queues with exchange of work (chunks of the loop iteration) if required. The underlying assumption of affinity scheduling algorithms remains that each execution of a loop is independent of previous executions.

In 1998 Bull [2] proposed a different loop scheduling algorithm, called Feedback Guided Dynamic Loop Scheduling (FGDLS). This method investigates a structure of loops where the parallel loop is contained within a surrounding sequential loop. Another assumption is that the workload of parallel loops does not change or changes only slowly from one execution to the next, so that observed timing information from the current execution of the loop can be used to guide the scheduling of the next execution of the same loop. In this way it is possible to limit the loss of performance associated with guided self-scheduling algorithms and the synchronization costs of affinity scheduling algorithms and to achieve an optimal load balance. Moreover, studies have shown that the FGDLS method achieves stability in the first few iterations so that there is no scheduling cost for the rest of parallel loops. Now, several Open MP compilers [1] have

*Supported by the Boole Centre for Research in Informatics, UCC
†Corresponding Author

used the method to detect and schedule this particular type of loops.

A important direction of this research deals with developing efficient FGDLS algorithms. Bull [2] gave a description of the method but no formal study of it was offered. Tabirca et.al. [11] started from Bull's solution and introduced an $O(\log p)$ algorithm called Continuous FGDLS. This first algorithm uses the execution times to generate a partial workload function that is split into equal chunks by using integrals to find the next scheduling bounds. A more natural approach was considered where the partial workload is a discrete sequence which is split into equal chunks by using sums [12]. This solution, named Discrete FGDLS finds the scheduling bounds in $O(p + \log p)$.

2 Feedback Guided Dynamic Loop Scheduling Method

The problem to be considered is the scheduling, across p processors $P_1, P_2, ..., P_p$, of the sequence of parallel loops that range from 1 to n (see [2] and [3]).

```
do seq t=1,nsteps
   do par i=1,n
      call loop_body(i)
   end do
end do
```

The FGDLS method uses a block scheduling that is given by the lower and upper loop iteration bounds $\{l_j^t, h_j^t \in N,\ j = 1, 2, \ldots, p\}$ on outer iteration (outer *sequential* iteration) t: where N are the natural numbers. These bounds satisfy the equation

$$l_1^t = 1,\ h_p^t = n,\ l_{j+1}^t = h_j^t + 1,\ j = 1, 2, \ldots, p-1.$$

We also assume that the corresponding execution times on the outer iteration t are measured and are given by $\{T_j^t, j = 1, 2, \ldots, p\}$. Based on these execution times, a piecewise constant approximation to the workload at outer iteration t is given by

$$\hat{w}_i^t = \frac{T_j^t}{h_j^t - l_j^t + 1},\ \ l_j^t \le i \le h_j^t,\ \ j = 1, 2, \ldots, p.$$

Note that $\hat{w}_i^t$ can be seen as the mean observed workload per loop iteration index on outer iteration t. The FGDLS method defines the new iteration bounds $\{l_j^{t+1}, h_j^{t+1} \in N, j = 1, 2, \ldots, p\}$ for the outer iteration $t+1$ so that this piecewise constant function is approximately equipartitioned amongst the p processors:

$$\sum_{i=l_j^{t+1}}^{h_j^{t+1}} \hat{w}_i^t \approx \frac{1}{p}\sum_{i=1}^{n} \hat{w}_i^t = \frac{1}{p}\sum_{k=1}^{p} T_k^t,\ \ j = 1, 2, \ldots, p. \quad (1)$$

We assume the the workload of loop_body(i) is represented by the workload $\{w_i,\ i = 1, 2, ..., n$ for which no information is known.

2.1 Discrete FGDLS

The Discrete FGDLS approach considers the execution times as sums of the iteration workloads so that

$$T_j^t = \sum_{k=l_j^t}^{h_j^t} w_k,\ j = 1, 2, ..., p.$$

In this case the quantity $\frac{1}{p}\sum_{k=1}^{p} T_k^t = \frac{1}{p}\sum_{i=1}^{n} w_i = \overline{W}$ from Equation (1) is an invariant for the method. Tabirca et al. [12] proposed a method based on the partial sums of the piecewise constant workloads

$$f^t(i) = \sum_{k=1}^{i} \hat{w}_k^t,\ i = 1, 2, ..., n \quad (2)$$

and the corresponding f^t-inferior part function by

$$f_{[]}^t(x) = i \Leftrightarrow f^t(i) \le x < f^t(i+1). \quad (3)$$

Based on these functions and the equation of Discrete FGDLS upper bounds $\{h_j^{t+1},\ j = 1, ..., p\}$ are given by (see [12])

$$h_0^{t+1} = 0,\ h_j^{t+1} = f_{[]}^t\left(f^t(h_{j-1}^{t+1}) + \overline{W}\right),\ j = 1, ..., p. \quad (4)$$

The functions f^t and $f_{[]}^t$ can be expressed by the execution times as follows:

$$l_j^t \le i \le h_j^t \Rightarrow f^t(i) = \sum_{q=1}^{j-1} T_q^t + \frac{T_j^t}{h_j^t - l_j^t + 1} \cdot (i - l_j^t + 1). \quad (5)$$

and

$$f^t\left(h_{j-1}^t\right) < x \le f^t\left(h_j^t\right) \Rightarrow$$

$$f_{[]}^t(x) = h_{j-1}^t + \left[\left(x - \sum_{q=1}^{j-1} T_q^t\right) \cdot \frac{h_j^t - l_j^t + 1}{T_j^t}\right]. \quad (6)$$

An $O(p + \log p)$ algorithm was derived from these equations [12] which achieves stability of the upper bounds ($h_j^t = h_j^{t+1}, \forall t > t_0$) in fewer iterations. The main inconvenience of this approach is that the execution times are not optimally balanced because the last processor always gets a load which is bigger than $\overline{W}$. Another drawback of this solution is that the bound h_j^{t+1} depends on the bound h_{j-1}^{t+1} as Equation (4) shows. This means that the computation of the upper bounds $\{h_j^{t+1},\ j = 1, 2, ..., p\}$ is inherently sequential, so that the potential of parallel computation cannot be used to calculate them.

3 A New Approach for Discrete FGDLS

This new approach introduces a very clever way to overcome the inconvenience outlined above. The starting point is the fundamental equation of FGDLS (1) that can be transformed as follows

$$\sum_{i=l_j^{t+1}}^{h_j^{t+1}} \hat{w}_i^t \approx \frac{1}{p}\sum_{k=1}^{p} T_k^t = \overline{W},\ \ j=1,2,\ldots,p \Leftrightarrow$$

$$\sum_{i=1}^{h_j^{t+1}} \hat{w}_i^t \approx \frac{j}{p}\sum_{k=1}^{p} T_k^t = j\cdot\overline{W},\ \ j=1,2,\ldots,p. \quad (7)$$

The approximation of Equation (7) can be done in the same way as in [12] by using

$$\sum_{i=1}^{h_j^{t+1}} \hat{w}_i^t \le j\cdot\overline{W} < \sum_{i=1}^{h_j^{t+1}+1} \hat{w}_i^t,\ \ j=1,2,\ldots,p, \quad (8)$$

which can be re-written as

$$f^t\left(h_j^{t+1}\right) \le j\cdot\overline{W} < f^t\left(h_j^{t+1}+1\right),\ \ j=1,2,\ldots,p, \Leftrightarrow \quad (9)$$

$$h_j^{t+1} = f_{[]}^t\left(j\cdot\overline{W}\right),\ \ j=1,2,\ldots,p. \quad (10)$$

The new equation (10) of the bound h_j^{t+1} is more convenient that Equation (4) since contains the feedback information from the step t represented by $f_{[]}^t$ and the constant quantity $j\cdot\overline{W}$. Moreover, based on Equation (6) it gives the following direct computation of the bound

$$\sum_{q=1}^{u(j)-1} T_q^t < j\cdot\overline{W} \le \sum_{q=1}^{u(j)} T_q^t \Rightarrow$$

$$h_j^{t+1} = h_{u(j)-1}^t + \left[\left(j\overline{W} - \sum_{q=1}^{u(j)-1} T_q^t\right)\frac{h_{u(j)}^t - h_{u(j)-1)}^t}{T_j^t}\right]. \quad (11)$$

Hence, all the upper bounds $h^{t+1} = \{h_j^{t+1},\ j=1,2,...,n\}$ for the parallel loop $t+1$ can be found from the bounds $h^t = \{h_j^t,\ j=1,2,...,n\}$ and the execution times $T^t = \{T_j^t,\ j=1,2,...,n\}$ using the parallel computation describe in Fig.1.

A prefix tree parallel computation can be employed to calculate the sequence of partial sums $S_j = \sum_{i=1}^{j} T_i^t, j = 1,2,...,p$ in a complexity of $O(\log p)$ [6]. The index $u(j)$ so that $S_{u(j)-1} < j\cdot\overline{W} \le S_{u(j)}$ can be also found in $O(\log p)$ using a sequential binary search. Therefore, we can conclude that the complexity of the parallel procedure DiscreteFGDLS is $O(\log p)$.

```
Inputs:
    n - the number of parallel loops
    p - the number of processors
    $T^t = \{T_j^t,\ j=1,2,..,p\}$ - the execution times
    $h^t = \{h_j^t,\ j=1,...,p\}$ - the upper bounds at $t$.
Output:
    $h^{t+1} = \{h_j^{t+1},\ j=1,...,p\}$ - the new upper bounds.
procedure DiscreteFGDLS($p, n, T^t, h^t, h^{t+1}$)
begin
    compute in parallel $S_j = \sum_{i=1}^{j} T_i^t, j=1,...,p$;
    $\overline{W} \leftarrow \frac{S_p}{p}$; $x_0^{t+1} \leftarrow 0$;
    do par j=1,p
        Find $u(j)$ so that $S_{u(j)-1} < j\cdot\overline{W} \le S_{u(j)}$.
        $h_j^{t+1} = h_{u(j)-1}^t +$
            $+ \left[\left(j\cdot\overline{W} - S_{u(j)-1}\right)\cdot\frac{h_{u(j)}^t - h_{u(j)-1}^t}{T_j^t}\right]$
    end do
end
```

Figure 1. An $O(\log p)$ Algorithm for Discrete FGDLS.

3.1 Fully Covering Property

Equation (1) gives an balanced distribution of the workloads $\{\hat{w}_i^t,\ i=1,2,...,n\}$ onto the processors. The extreme case to avoid is when there are processors that are not scheduled any iterations. For example if the piecewise constant workloads are $\{1,1,1,1,11\}$ to be scheduled on 3 processors, the average $\overline{W}$ becomes $\overline{W} = \frac{1+1+1+1+11}{3} = 5$. Equation (1) gives that the first 4 iterations are assigned to processor 1 and the last one to processor 2 hence processor 3 does not get any iteration. The case when all the processors receive iterations is called *Fully Covering* and it happens if the upper bounds $\{h_j^t,\ j=1,2,...,p\}$ satisfy

$$h_1^t < h_2^t < ... < h_p^t = n. \quad (12)$$

For the Discrete and Continuous FGDLS algorithms (see [11] [12]) the fully covering property was proven when the initial workloads $\{w_i,\ i=1,2,...,n\}$ satisfy

$$w_i \le \frac{1}{p}\cdot\sum_{l=1}^{n} w_l = \overline{W},\ i=1,2,...,n. \quad (13)$$

This property means that no workload is expected to be bigger that the average $\overline{W}$. We also prove that the above algorithm achieves fully covering when Equation (13) holds.

Theorem 1 *If the initial workloads $\{w_i,\ i=1,2,...,n\}$ satisfy Equation (13), then the piecewise constant work-*

loads $\{w_i^t,\ i=1,2,...,n\}$ *satisfy*

$$w_i^t \le \frac{1}{p}\cdot\sum_{l=1}^{n} w_l^t = \frac{1}{p}\cdot\sum_{l=1}^{n} w_l = \overline{W},\ i=1,2,...,n. \quad (14)$$

Proof. Let w_i^t be a piecewise constant workload with $l_j^t \le i \le h_j^t$.
Starting from the definition of w_i^t we find

$$w_i^t = \frac{T_j^t}{h_j^t - l_j^t + 1} = \frac{\sum_{q=l_j^t}^{h_j^t} w_q}{h_j^t - l_j^t + 1} \le$$

$$\frac{\sum_{q=l_j^t}^{h_j^t} \max_{l=1}^{n} w_l}{h_j^t - l_j^t + 1} = \max_{l=1}^{n} w_l.$$

Since $\max_{l=1}^{n} w_l$ is an element of the sequence $\{w_i,\ i=1,2,...,n\}$ it follows that $\max_{l=1}^{n} w_l \le \frac{1}{p}\cdot\sum_{l=1}^{n} w_l$ therefore we have

$$w_i^t \le \max_{l=1}^{n} w_l \le \frac{1}{p}\cdot\sum_{l=1}^{n} w_l = \frac{1}{p}\cdot\sum_{l=1}^{n} w_l^t.$$

♠

Theorem 2 *If the initial workloads* $\{w_i,\ i=1,2,...,n\}$ *satisfy Equation (13), then the upper bounds* $\{h_j^{t+1},\ j=1,2,...,p\}$ *computed by procedure DiscreteFGDLS satisfy*

$$h_1^{t+1} < h_2^{t+1} < ... < h_p^{t+1} = n. \quad (15)$$

Proof. Equation (9) gives that h_j^{t+1} is the biggest index h so that $f^t(h) \le j\cdot\overline{W}$. We start from $f^t\left(h_j^{t+1}+1\right) = f^t\left(h_j^{t+1}\right) + \hat{w}^t_{h_j^{t+1}+1}$ which can be transformed using $f^t\left(h_j^{t+1}\right) \le j\cdot\overline{W}$ and $\hat{w}^t_{h_j^{t+1}+1} < \overline{W}$ into $f^t\left(h_j^{t+1}+1\right) \le (j+1)\cdot\overline{W}$.
Since h_{j+1}^{t+1} is the biggest index h to give $f^t(h) \le (j+1)\cdot\overline{W}$ we find that $h_j^{t+1}+1 \le h_{j+1}^{t+1}$ which is equivalent with $h_j^{t+1} < h_{j+1}^{t+1}$. So the new upper bounds strictly increase. ♠

4 Practical Experiments

Some experiments have been carried out in order to measure the performance of the new Discrete FGDLS algorithm and to compare it against the other Discrete FGDLS. These experiments have considered theoretical execution times given by summation of workloads since this gives an accurate evaluation of the model.

FGDLS uses a block partition to schedule the parallel loops at each outer iteration so that we can consider that this is given by $\{(l_j^{d1,t}, h_j^{d1,t}),\ j=1,2,...,p\}$ for the previous Discrete FGDLS algorithm and $\{(l_j^{d2,t}, h_j^{d2,t}),\ j=1,2,...,p\}$ for this new Discrete FGDLS solution. These two block schedulings give the following theoretical execution times per processors: $T_j^{d1,t} = \sum_{l_j^{d1,t}}^{h_j^{d1,t}} w_i$ and $T_j^{d2,t} = \sum_{l_j^{d2,t}}^{h_j^{d2,t}} w_i$.

The experiments consider the loop structure

```
do seq t=1,1000
   do par i=1,1000
       call loop_body(i)
   end do
end do
```

where the procedure loop_body(i) computes w_i times a floating point operation.
For the workloads $\{w_i = 1000 - i,\ i=1,2,...,1000\}$ and $p=4$ processors the variation of the bounds $\{h_j^{d1,t},\ j=1,2,3,4\}$ and $\{h_j^{d2,t},\ j=1,2,3,4\}$ is given by

```
  Old Discrete FGDLS      |  New Discrete FGDLS
 t=1 250 500 750 1000     |  250 500 750 1000
 t=2 142 298 498 1000     |  142 300 500 1000
 t=3 134 292 497 1000     |  134 293 500 1000
 t=4 134 293 502 1000     |  134 293 500 1000
 t=5 134 292 499 1000     |  134 293 500 1000
 t=6 134 292 499 1000     |  134 293 500 1000
```

and generates the following execution times $\{T_j^{d1,t},\ j=1,2,3,4\}$ and $\{T_j^{d2,t},\ j=1,2,3,4\}$.

```
      Old Discrete FGDLS
t=1   218,875 156,375  93,875  31,375
t=2   131,989 121,758 120,500 126,253
t=3   125,089 124,425 124,230 126,756
t=4   125,089 125,133 126,027 124,251
t=5   125,089 124,425 125,235 125,751
t=6   125,089 124,425 125,235 125,751
      New Discrete FGDLS
t=1   218,875 156,375  93,875  31,375
t=2   131,989 123,161 120,100 125,250
t=3   125,089 125,133 125,028 125,250
t=4   125,089 125,133 125,028 125,250
t=5   125,089 125,133 125,028 125,250
t=6   125,089 125,133 125,028 125,250
```

From these execution times one can see that the new Discrete FGDLS method achieves a better load balance than the old Discrete FGDLS. Moreover, the stability of the upper bounds is reached in fewer steps. Similar tests were done for several types of workloads for example polynomials, uniformly distributed, normally distributed etc. We found that the new Discrete FGDLS algorithm always achieves a better load balance than the previous Discrete FGDLS algorithm.

The above loop structure has been also implemented and run on an Origin 3000 shared memory machine using the same type of initial workloads. Similar findings have been observed for the parallel computation with $p = 4$ processors. The old Discrete FGDLS algorithm (FGDLS1) has stabilised after 5 iterations while the new Discrete FGDLS algorithm (FGDLS2) has reached stable bounds after 3 iterations.

Table 1 and Figure 2 present the running times for the first parallel loops $t = 1, 2, ..., 6$ which were measured in milliseconds. It shows that from the sixth parallel loop on the new Discrete FGDLS algorithm (FGDLS2) is 0.2 milliseconds more efficient than the old one (FGDLS1). Since this is repeated almost 1000 times it results an important difference for the whole parallel loop structure. Table 2 and Figure 3 illustrate how the overall running times in seconds reduces when the number of processors increases. These times came very close with a marginal advantage for the new Discrete FGDLS method.

t	1	2	3	4	5	6
FGDLS1	17.3	15.6	14.5	13.8	13.3	13.3
FGDLS2	17.2	14.3	13.7	13.1	13.1	13.1

Table 1. Processor Times for $t = 1, ..., 6$.

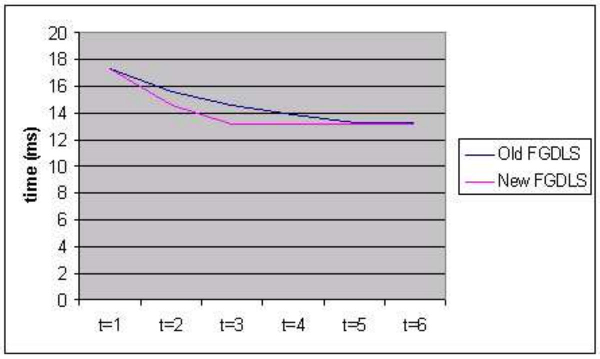

Figure 2. Processor Times for $t = 1, ..., 6$.

5 Conclusions

This article has proposed an $O(\log p)$ solution for the Discrete FGDLS method. A condition has been proposed for the new algorithm to achieve the fully covering property. The practical comparison has found that the new Discrete FGDLS solution produces better load balance than the previous Discrete FGDLS for a variety of workloads. It has also observed that this new solution reaches stability quicker than the previous solution. These arguments demonstrate that the new algorithm is the most efficient solution for FGDLS.

	$p = 1$	$p = 2$	$p = 4$	$p = 8$
FGDLS1	52.78	27.65	13.57	7.45
FGDLS2	52.79	26.89	13.23	7.12

Table 2. Execution Times for $p = 1, 2, 4, 8$.

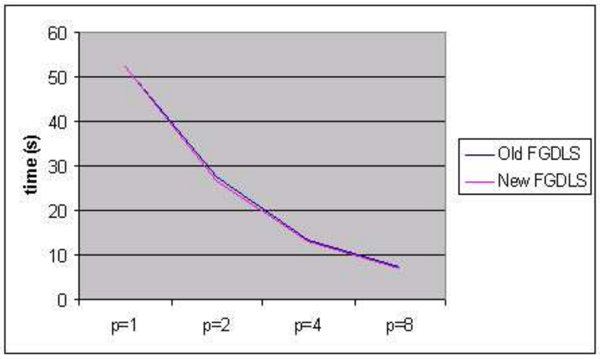

Figure 3. Execution Times for $p = 1, 2, 4, 8$.

References

[1] Ayguade E., Blainey B., Duran A., Labarta J., Martinez F., Martorell X. and Silvera R.: Is the SCHEDULE Clause Really Necessary in OpenMP?, in Proceedings of IWOMP Applications and Tools, Springer-Verlag, vol. LNCS 2716 (2003) 69-83.

[2] Bull, J.M.: Feedback Guided Loop Scheduling: Algorithms and Experiments, in Proceedings of Euro-Par'98, Springer-Verlag, vol. LNCS 1470, (1998) 377-382.

[3] Bull, J.M., Ford, R.W., Freeman, T.L. and Hancock, A.: A Theoretical Investigation of Feedback Based Load Balance Algorithm, in e-Proceedings of Ninth SIAM Conference on Parallel Processing for Scientific Computing, SIAM Press (1999).

[4] Eager, D.L. and Zahorjan, J.: Adaptive Guided Self-Scheduling, Technical Report 92-01-01, Department of Computer Science and Engineering, University of Washington (1992).

[5] Hummel, S.F., Schonberg, E. and Flynn, L.E.: Factoring: A Practical and Robust Method for Scheduling Parallel Loops, Communications of the ACM, vol. 35, no. 8, (1992) 90–101.

[6] Jaja, J.: An Introduction to Parallel Algorithms, Addison-Wesley (1992).

IEEE COMPUTER SOCIETY

[7] Lucco, S.:A Dynamic Scheduling Method for Irregular Parallel Programs, in Proceedings of ACM SIGPLAN '92, San Francisco, CA, June 1992, (1985) 200–211.

[8] Markatos, E.P. and LeBlanc, T.J.:Using Processor Affinity in in Loop Scheduling on Shared Memory Multiprocessors, IEEE Transactions on Parallel and Distributed Systems, vol. 5, no. 4,(1994) 379–400.

[9] Polychronopoulos, C.D. and Kuck, D.J.: Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers, IEEE Transactions on Computers, vol. C-36, no. 12, (1987) 1425–1439.

[10] Subramanian, S. and Eager, D.L.: Affinity Scheduling of Unbalanced Workloads, in Proceedings of Supercomputing'94, IEEE Comp. Soc. Press, (1994) 214–226.

[11] Tabirca, T., Freeman, L. and Tabirca, S.: An O(log p) Parallel Algorithm for Feedback Guided Dynamic Loop Scheduling, Journal of Parallel Algorithms and Applications, 17, (2002) 157 - 164.

[12] Tabirca T., Tabirca S., Freeman L., Yang T.L.: An O(p+logp) Algorithm for the Discrete FDGLS, Proceedings of ICPP-HPSECA 2003, Taiwan, (2003) 164-173.

[13] Tzen, T.H. and Ni, L.M.: Trapezoid Self-Scheduling Scheme for Parallel Computers, IEEE Trans. on Parallel and Distributed Systems, vol. 4, no. 1, (1993) 87–98.

[14] Yan, Y., Jin, C. and Zhang, X.: Adaptively Scheduling Parallel Loops in Distributed Shared-Memory Systems, IEEE Trans. on Parallel and Distributed Systems, vol. 8, no. 1, (1997) 70–81.

IEEE COMPUTER SOCIETY