SIMPLE NEAR OPTIMAL PARTITIONING APPROACH TO PERFECT TRIANGULAR ITERATION SPACE

Nedal Kafri
Department of Computer Science
Al–Quds University
Palestine
Email: nkafri@science.alquds.edu

Jawad Abu Sbeih
Department of Computer Science
Al Quds Open University
Palestine
Email: jabusbeih@qou.edu

KEYWORDS

Parallel Computing, Nested Loops, Static Partitioning, Load Balancing.

ABSTRACT

One of the most critical issues in parallel computing is the efficient distribution of a workload and data (workload balancing) amongst networked processors in multiprocessor and multicomputer systems to achieve optimal performance. Vast large scale scientific computing, such as numerical and Digital Signal Processing (DSP) problems have the nested loops as the main parallelized code segment. The main concern in partitioning the iteration space is the trade off between load balance, data locality, and minimizing scheduling overheads. Therefore, it is important to study and implement efficient decomposition techniques, which play an important role in achieving optimal performance and efficient use of multiprocessor and multicomputer systems. In this work, we focus on static decomposition of perfect triangular iteration space to achieve load balancing across given processors in a homogenous system. This paper introduces an intuitive near-optimal partitioning approach to triangular iteration space of a loop nest along the outermost loop index. Furthermore, the obtained partitions thus consist of contiguous non-overlapping parts which preserve data locality.

INTRODUCTION

Parallel computing is an efficient technique used to achieve high performance and efficient use of multiprocessor and multicomputer systems which speedup many applications over sequential processing on a single processor. The major goal of parallelization is to minimize the *elapsed time* (ie., overall computation time) by distributing the computation workload amongst the available processors evenly. This distribution can be done automatically by compilers (Chaundhary et al., 1996; Haghighat and Polychronopoulos, 1996; Hudak and Abraham, 1990; Jialin, 1998; Petkov et al., 2002), or manually by programmers using compiler directives and other different *decomposition* programming techniques (Fahringer, 1998; Hancock et al., 2000b; Kejariwal et al., 2005; Li et al., 2000; Sakellariou, 1996).

Significant amount of applications such as numerical, large scale scientific problems, Digital Signal Processing (DSP) (Li et al., 2000), computer vision, high-definition television medical imaging, remote sensing and many cryptographic algorithms, such as unchained Skipijak and DES (Petkov et al., 2002), are considered to be multi-dimensional problems. It is generally agreed by researchers that most of the computation time is spent in loops, which can be single (one-dimensional) or nested ones (multidimensional). In the case of nested loops, the iteration space (number of iterations of the loop body or workload) of these loops can be of constant, or iteratively either increasing or decreasing size. Thus, nested loops (loop nest) are the most important portion of these applications. Therefore, most researchers pay much of their attention to loop parallelization (D'Hollander, 1992; Chaundhary et al., 1996; Haghighat and Polychronopoulos, 1996; Hancock et al., 2000b; Hudak and Abraham, 1990; Jialin, 1998; Kejariwal et al., 2004, 2005; Li et al., 2000; Petkov et al., 2002; Polychronopoulos et al., 1986; Sakellariou, 1996; Xue et al., 2005).

Efficient parallel execution of these applications requires efficient *partitioning* and *mapping* of the iteration spaces of these nested loops across available processors to achieve perfect load balance. Therefore, it is important to study effective mapping techniques to gain significant speedup from parallelization. Thus, mapping of loop nests have received extensive attention in literature; mapping of loop nests with rectangular iteration spaces has received coverage in (D'Hollander, 1992; Polychronopoulos et al., 1986), whereas partitioning of loop nests with non-rectangular iteration space has been covered in (Haghighat and Polychronopoulos, 1996; Kejariwal et al., 2005; Sakellariou, 1996). However, (Haghighat and Polychronopoulos, 1996; Sakellariou, 1996) do not partition the iteration space uniformly across different processors and have several limitations, such as trade-off between *parallelism* and *data locality*. Furthermore, these approaches do not address the problem of partitioning iteration spaces with variable densities, i.e., loops with non-constant strides. Whereas in

Proceedings of the 2008 High Performance
Computing & Simulation Conference ©ECMS
Waleed W. Smari (Ed.)
ISBN: 978-0-9553018-7-2 / ISBN: 978-0-9553018-6-5 (CD)

(Kejariwal et al., 2005), they address the distribution of iteration spaces with variable densities based on geometric approach for computing the iteration space before mapping.

Based on whether the workload is distributed before or during run-time, loop partitioning can be classified as *static*, (where, usually partitioning is the most important aspect) or *dynamic* (Hancock at al., 2000a; Hancock et al., 2000b) (where, usually scheduling is the most important aspect), respectively. It should be noted that dynamic scheduling techniques may require additional communication and (runtime) overheads to achieve the load balanc. Furthermore, the static load balancing on *heterogeneous* systems, where the processors have different speeds and capacities is discussed in (Beaumont et al., 2002).

This paper focuses on *static* partitioning of loop nest with perfect *triangular* iteration space in *homogenous platforms*. It introduces an approximation–based approach that partitions the iteration space along the axis corresponding to the index of the outermost loop to achieve near optimal load balancing. The analytical and experimental results show that the proposed approach competes the best known techniques by its simplicity. Furthermore, the partition thus obtained consists of contiguous and disjoint subsets, which facilitates exploitation of data locality.

The rest of this paper is organized as follows: The next section presents a background relevant to the loop nest iteration space, partitioning problem, the terminology, notions, and definitions used in this paper. Thereafter, we introduce a new *approximation–based near optimal partitioning* approach to perfect triangular iteration space (ANOP). Next to that section, evaluation and experimental results are presented. Finally, we conclude and propose directions for future research.

BACKGROUND

In some applications, the workload balancing of the iteration space is primitive. This can be achieved by dividing the iteration space among P processors evenly into P parallel tasks (possibly into embarrassingly or trivially parallel tasks i.e., tasks that can be done independently of any other computation, without any communication among them). In other words, the load balance can be achieved by the decomposition of the iteration space into a collection of equivalent disjoint subsets (parts) of the iteration space, whose union is all of the iteration space (Scott et al., 2005). To illustrate this definition, consider a very simple summation example.

$$A = \sum_{i=1}^{N} a_i$$

This kind of operation is called reduction; it reduces the vector $(a_1, a_2, \ldots, a_N)$ to the scalar A. Assume for simplicity that P divides N ($P|N$) i.e., N is an integer multiple, c, of P processors: $N = c.P$. Then we can divide the reduction operation into P disjoints partial sums:

$$A_k = \sum_{i=(k-1)c+1}^{ck} a_i, \quad for\, k = 1 \ldots P,\, Then \tag{1}$$

$$A = \sum_{k=1}^{P} A_k \tag{2}$$

Considering Equation (1) and (2), we have managed to create P embarrassingly parallel tasks, having $c = N/P$ addition operations (workload) to do on c data points.

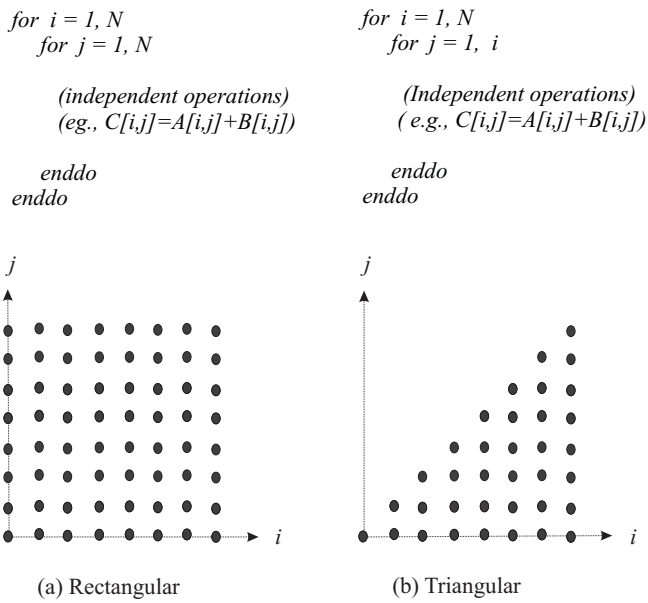
Similarly, one can use this simple decomposition technique of the workload of loop nests having a rectangular structure. Such loop nests and their iteration spaces can be represented by the pseudocode and its representation as a polytope in Figure 1(a). This geometric shape approach i.e., polytope representation, which has been used often since the early years of Lamport's hyperplane method (Lamport, 1974), allows us to deal with the problem from a geometric point of view hoping to provide a clearer understanding. Partitioning an iteration space along an axis corresponding to the outermost loop can achieve a perfect workload balance. More detailed discussion on rectangular loop partitioning techniques can be found in (D'Hollander, 1992; Polychronopoulos et al., 1986; Sakellariou, 1996).

However, those simple decomposition approaches in parallelizing cannot lead to workload balancing in many problems containing *non-rectangular* nested loops, such as *triangular* loop nests Figure 1(b). Vast number of triangular loop nests can be found in some matrix operations, for example, adding lower/upper triangular matrices, *LU* factorization problems, and prime numbers discovery.

Triangular Loop Nests

Triangular loop nests (or triangular iteration space) means a loop nest consisting of an outer loop and an inner loop having bounds dependent on the index of the outer loop. Moreover, the operation(s) in the inner loop can be done independently of all others. Such triangular loop nests and their iteration spaces can be represented by the pseudocode and its representation as a polytope in Figure 1(b) (Haghighat and Polychronopoulos, 1996; Kejariwal et al., 2005; Sakellariou, 1996), where dots indicate the number of iterations of the inner loop.

It should be noted that the operation(s) in the inner loop can be as simple as shown in Figure 1(b), which adds two lower triangular matrices, constant sequence of operations or a multidimensional/multilevel independent nested loops with constant iteration spaces i.e., invariant iterator (Kejariwal et al., 2005).



(a) Rectangular (b) Triangular

Figure 1: An Example to Illustrate Iteration Space of Loop Nests (a) Rectangular (b) Triangular

It is known that the total workload W (loop iterations) of such triangular loop nest shown in Figure 1(b) is equivalent to the sigma notation in Equation (3), where c represents the constant number of operations in the inner loop body (which can be assumed 1 for simplicity), and l and u are the lower and upper bounds of the outermost loop (i.e., along the i axis), respectively.

$$W = \sum_{i=l=1}^{u=N} \sum_{j=1}^{i} c = c \sum_{i=1}^{N} i = c\frac{N(N+1)}{2} \tag{3}$$

Detailed work on nested loops and computing loop iterations can be found in (Sakellariou, 1996).

Static Load Partitioning and Load Balancing

Partitioning the workload amongst processors of a multiprocessors or a multicomputer system plays a critical role in parallel processing. One of the requirements for decomposition is that the workload to be balanced across all the processors. If the work is not distributed equally, then one processor may end up taking longer time than the others. Since we are doing a cooperative project, the overall job cannot be accomplished until the slowest subtask is finished. Thus, Workload balancing is one of the critical issues and goals which plays an important role in parallel computing. Efficient and perfect distribution of the workload will enable to achieve this goal, and consequently will affect the overall performance and the efficiency of the parallel processing. The author in (Sakellariou, 1996) discusses the necessary conditions for partitioning a loop nest into *equal* workload. It should be noted that this work discusses static workload partitioning on *homogenous* systems, where the processors have *equal* speeds and capacities.

Assume that a set of P parallel tasks (indexed by $k = 1, \ldots, P$), executes its assigned workload W_k in an amount of time t_k. Furthermore, assuming that W'_k represents the set of those operations whose workload is W_k, then $W' = \cup_{k=1}^{p} W'_k$ and, for any two subsets, $W'_i, W'_j, 1 \leq i, j, \leq P, i \neq j$, it must be the case that $W'_i \cap W'j = \theta$. Consequently, it is clear that $W = \sum_{k=1}^{p} W_k$.

In (Scott et al., 2005) they define the average execution time and the load balance in term of time. Since we rarely predict the exact execution time in advance which is almost a factor of the workload in homogenous platform (i.e., $t_k = cW_k$), whereas we can frequently predict the workload in advance, we can redefine the average workload:

$$ave\{W_k : 1 \leq k \leq P\} = \frac{1}{P} \sum_{k=1}^{P} W_k$$

Furthermore, one can define the load balance β, resulting from a predetermined decomposition strategy, to be the ratio of the average workload to the maximum workload (i.e., to the biggest task):

$$\beta = \frac{ave\{W_k : 1 \leq k \leq P\}}{max\{W_k : 1 \leq k \leq P\}} \tag{4}$$

Therefore; a set of tasks is said to be load balanced if β is closed to one. The *optimal* (*perfect*) load balance W_{opt} is achieved if, for all $k, 1 \leq k \leq P$,

$$W_k = \frac{W}{P} = W_{opt} \tag{5}$$

in this case, β equals one.

On the other hand, inefficient distribution leads to *load imbalance* D. Whenever there is a process k for which the *difference* or *deviation* $D_k = W_k - W/p$ is non-zero, then the workload assigned to *k-th* processor exhibits an imbalance equal to D_k. Negative value of D_k means that *k-th* processor is assigned less workload than the average share, while positive value of D_k indicates that processor k is assigned more workload than the average workload and can be considered *'overloaded'*. As a result of this overload, the latter processor determines the overall parallel computation time. Consequently, the exhibited load imbalance, D, resulting from a given distribution of the workload W amongst P processors (Sakellariou, 1996), given by

$$D = \max_{1 \leq k \leq p} (D_k) = \max_{1 \leq k \leq p} (W_k - \frac{W}{p}) = W_{max} - \frac{W}{p} \tag{6}$$

In order to assess the impact of the *load imbalance* D on the overall parallel elapsed time, Sakellariou in (Sakellariou, 1996) introduces a *relative load imbalance* D_R:

$$D_R = \frac{D}{W_{max}} = \frac{W_{max} - \frac{W}{P}}{W_{max}} = 1 - \frac{W}{PW_{max}} \qquad (7)$$

It is clear that D_R takes values in the interval $[0, 1 - 1/p]$. Values close to zero denote perfect load balance and a closed to linear-speedup is obtained. Whereas, non-zero values denote load imbalance, values closed to $1 - 1/p$ denote highly imbalanced workload decomposition, and no significant speedup from parallelization can be expected. Thus, in order to increase the gains from parallelization, the workload requires distribution of the workload as evenly as possible. Therefore, we must search for a strategy for mapping the workload such that the *relative load imbalance* will be close to zero, and the load balance β will be close to one.

In order to achieve static workload balance in a homogenous architecture, several approaches were introduced in the literature. The most common approaches for static loop partitioning on homogenous systems are:

- Block Partitioning (BP) (Kruskal and Weiss, 1985): A simple way to distribute the iteration space (workload) of a single loop or a rectangular loop nest is *block partitioning* BP. Assume that P is the number of processors and N is the iteration determined by the outer loop, where $(P|N)$, one can gain perfect workload partitioning using this technique. Block partitioning distributes contiguous equal iterations onto processors in a consecutive manner, which in turn preserve *data locality*. Thus, Processor 1 executes iterations 1 through N/P. Process 2 executes iterations $N/P + 1$ through $2N/P$. In general the *k-th* processor, for all $k, 1 \leq k \leq P$ executes iterations $((k-1)N/P + 1)$ through (kN/P).

- Cyclic Partitioning (CP): The second common distribution approach that can be used to distribute loop workload is called *cyclic partitioning* CP or *stride mapping*. Stride technique is a widely used technique, to decompose the loop iterations as evenly as possible with each process being assigned a fixed number of iterations of the outer loop in a round robin fashion; process 1 executes iterations 1, $P+1$, $2P$ + 1,.., process 2 executes iterations 2, P+2, 2P+2, ... In general, the *k-th* processor, for all $k, 1 \leq k \leq P$ executes iterations $k + iP$, $i = 0, 1, 2, \ldots, (N/P - 1)$. Similar to BP approach, the partitions of a rectangular iteration space thus obtained consist of equal workload. However, CP generates a fragmented partition (i.e. each individual set is a collection of non-contiguous subsets). Consequently, this approach may exhibit poor performance in many applications due to *false sharing* (a well known phenomenon in computer architecture).

  With *triangular loop nest* partitioning, (BP) and (CP) approaches are no longer sufficient; These approaches may generate *unequal* parts. More efficient approaches for partitioning triangular iteration space consisting of independent loops are:

- Balanced Chunk Scheduling (BCS) (Haghighat and Polychronopoulos, 1993): Unlike block and cyclic partitioning which distribute only the iterations of the outer loop, BCS attempts to partition the total number of iterations of the loop nest body among processors as evenly as possible. An example of the latter is shown in Appendix B of (Sakellariou, 1996). However, Haghighat and Polychronopolous restrict their discussion to double loop. Moreover, requires predetermination of the total number of index points in the iteration space.

- Canonical Loop Partitioning (CLP) (Sakellariou, 1996): Sakellariou introduces a notion of *canonical loop nest* for loop mapping. CLP assumes that the outermost loop can be partitioned into $2P^{m-1}$ equal parts, where P is the number of processors, and m is the depth of a loop nest. However, this may generate empty subsets which lead to a load imbalance. Moreover, CLP generates a non-contiguous partition. CLP employs an enumeration-based approach to determine the total number of index points in an iteration space. It relies on loop normalization in the presence of non-unit strides. However, the introduction of floors and ceilings renders this approach nonviable in practice. Furthermore, determination of the set boundaries in CLP is very cumbersome.

- Weight-Based Partitioning WBP (Kejariwal et al., 2005): This approach discusses the partitioning of loop nest with N-dimensional non rectangular iteration space, with variable densities following a geometric approach. Based on the assumption that there do not exists any invariant iterator, it introduces a procedure for partitioning an iteration space. The procedure consists of three major steps. First, it computes a partial weight of the convex polytope as a function of the outermost index variable. They follow a weight-based approach for estimating the number of index points in a polytope. Next, the algorithm computes the total weight of the convex polytope corresponding to loop nest. Finally, it determines the breakpoints along an axis corresponding to the outermost index. WBP algorithm achieves *near-optimal* and *contiguous* partitions of an iteration space.

APPROXIMATION–BASED PARTITIONING

This work focuses on a perfect loop nest of depth 2 as shown in Figure 1(b), where the bound of the inner loop

depends on the index of the outer loop. We attempt to discover an efficient, intuitive, and simple approach to distribute total iteration space among P processors as evenly as possible (i.e., optimizing the load balance), so that the load balance β is maximized, and the load imbalance D as well as the relative load imbalance D_R are minimized as a result.

For simplicity, assume that the constant $c = 1$ (in Equation 3), our goal becomes equivalent to finding the optimal workload (Equation 5) for all $k, 1 \leq k \leq P$ such that

$$W_k = W_{opt} = \frac{W}{P} = \frac{N(N+1)}{2P}$$

Now, Consider Figure 1(b) and the corresponding workload in Equation (3). Partitioning such loop nest (along the axis corresponding to the outermost loop, where the lower bound $l = 1$ and the upper bound $u = N$ into equi-workload (E_k) across P processors, is equivalent to discovering the perfect lower bound l_k and the upper bound u_k of the outer loop assigned to the *k-th* processor (i.e., the *k-th* partition), for all $k, 1 \leq k \leq P$, such that

$$|E_k - \frac{W}{P}|$$

is minimized (Sakellariou, 1996), where

$$\sum_{i=l=1}^{u=N} \sum_{j=1}^{i} 1 = \sum_{k=1}^{P} \sum_{i=l_k}^{u_k} \sum_{j=1}^{i} 1 = \sum_{k=1}^{P} W_k$$

$l_1 = l = 1, u_p = u = N$

$$E_1 = \sum_{i=l_1}^{u_1} i = \frac{u_1(u_1+1)}{2} \tag{8}$$

and, for all $2 \leq k \leq P, l_k = u_{k-1} + 1$

$$E_k = \sum_{i=l_k}^{u_k} \sum_{j=1}^{i} 1 = \frac{u_k(u_k+1)}{2} - \frac{u_{k-1}(u_{k-1}+1)}{2} \tag{9}$$

Recalling that $l_1 = l = 1$ and $l_k = u_{k-1} + 1$, for all $2 \leq k \leq P$, we can express the problem of finding the perfect load balance as that of finding integer u_k, $1 \leq k \leq P$, such that

$$E_k = \sum_{i=l_k}^{u_k} \sum_{j=1}^{i} 1 \approx \frac{W}{P}$$

Based on Equation (9), it is clear that

$$\sum_{j=1}^{k} E_j = \sum_{i=l_1}^{u_k} i = \frac{u_k(u_k+1)}{2}$$

and

$$\frac{k}{P} W = \frac{k}{P} \frac{N(N+1)}{2},$$

we can formulate the following approximation equation:

$$\sum_{j=1}^{k} E_j = \sum_{i=l_1}^{u_k} i = \frac{u_k(u_k+1)}{2} \approx \frac{k}{P} W = \frac{k}{P} \frac{N(N+1)}{2}$$

Consequently, we obtain

$$u_k^2 + u_k = \frac{k}{p} N^2 + \frac{k}{p} N \tag{10}$$

Assuming that the difference between the terms u_k and $\frac{k}{p}N$ in Equation (10) is very small (i.e., $u_k \approx \frac{k}{p}N$), it has no significant impact on the equation in front of the exponent terms (u_k^2 and $\frac{k}{p}N^2$). Therefore, they can be omitted from the equation, for the purpose of simplicity. Thus, the direct solution for the *k-th* upper bound u_k is

$$u_k = N\sqrt{\frac{k}{p}}$$

Clearly, it appears that integer solutions for u_k are not common; thus rounding u_k to nearest integer, the formula becomes:

$$u_k = round(N\sqrt{\frac{k}{p}}) \tag{11}$$

In the following section, we introduce an example using this approach with the gained analytical and experimental results.

EVALUATION AND EXPERIMENTAL RESULTS

To illustrate and evaluate the proposed partitioning approach, consider the code segment shown in Figure 1(b) for $N = 800$, which adds two upper triangular matrices (800x800 matrices). A similar example was provided in (Sakellariou, 1996)(sec. 2.3.1.1), to illustrate Balanced Chunk Scheduling originally approached in (Haghighat and Polychronopoulos, 1996). In order to partition the iteration space along the outer loop (i loop) across 8 processors, the iterations are distributed using the proposed approach following Equation (11). As shown in Table 1, processor 1 executes iteration $l_1 = 1$ through $u_1 = 283$, processor 2 executes iterations 284 through 400, and so on. Also, the table shows the corresponding workload E_k using Equations 8 and 9, the load deviation assigned to the *k-th* processor ($D_k = E_k - \frac{W}{P}$),

Table 1: Partitioning Loop Nest (for $N = 800$) Across 8 Processors, Corresponding Workload E_k, Workload Deviation D_k, and D_k/E_{max}

	Iteration Distribution			
P_k	$l_k - u_k$	E_k	D_k	D'_k
1	1-283	40186	136	0.0034
2	284-400	40014	-36	-0.0009
3	401-490	40095	45	0.0011
4	491-566	40166	116	0.0029
5	567-632	39567	-483	-0.0119
6	633-693	40443	393	0.0097
7	694-748	39655	-395	-0.0098
8	749-800	40274	224	0.0055

and the ratio of the *k-th* workload deviation to the maximum workload D'_k (where $D'_k = \frac{D_k}{E_{max}}$).

As a result of this distribution, the achieved average workload is equal to the perfect workload $\frac{W}{P} = 40050$, and the maximum assigned workload $W_{max} = W_6 = 40443$. Consequently, the exhibited load balance $\beta = 0.994438$ (equation 4) is close to 1, which indicates that the *near-optimal* partitioning of the workload W amongst P processors is achieved.

Similarly, it is clear that the resulting load imbalance is equal to ($D = 393$), according to Equation 6 and the *relative load imbalance* D_R using Equation 7 is equal to (0.0097), which can be considered close to zero.

It should be noted that when implementing our approach in the provided example in (Haghighat and Polychronopoulos, 1993) (for Balanced Chunk Scheduling BCS approach), as well as in the provided case study in (Kejariwal et al., 2005) for Weight-Based Partitioning WBP approach, it obtains same results.

In order to illustrate the *Weight-Based Partitioning* WBP technique for a triangular non-uniform iteration space (Kejariwal et al., 2005), they consider the well known *Sieve of Eratosthenes* (TSoE) algorithm that identifies all prime numbers up to a given number M . The following shows the kernel (loop nest) of the (TSoE) for $N = \sqrt{M}$:

```
doall i = 3, N, 2
   doall j = i, M, 2*i
        LOOP BODY
   end doall
end doall
```

To distribute the workload amongst P processors according to the WBP, we have to follow several steps: begin with checking the existence of an invariant iterator. If exists, then determine a partial weight of the convex polytope corresponding to the iteration space using a geometric approach. Next, determine the total weight of the convex polytope. Finally, the breakpoints (i.e., bounds) for the partitions can be determined. Table 2, shows a comparison of the determined upper bounds (u_k, for $1 \leq k \leq P$) (before rounding the results into integer numbers for comparison purpose) using WBP and our proposed (ANOP) mapping approaches for $M = 10000$ and $N = 32$ on different number of processors P.

Table 2: The Determined Upper Bounds (u_k) for Partitioning Loop Nest (of $N = 32$) Across 2 Processors, 3 Processors, and 4 Processors

	Determined Upper Bounds (u_k)	
	$P = 2$	
k	$ANOP$	WBP
1	22.63	22.66
2	32.00	32.00
	$P = 3$	
k	$ANOP$	WBP
1	18.48	18.57
2	26.13	26.14
3	32.00	32.00
	$P = 4$	
k	$ANOP$	WBP
1	16.00	16.13
2	22.63	22.67
3	27.71	27.72
4	32.00	32.00

The results of the provided examples show that the obtained partitions using our ANOP, the BCS, and the WBP mapping approaches are almost the same. Thus, we can expect equivalent performance of these mapping approaches and better performance than the BP and the CP techniques.

To analyze and evaluate the complexity of parallel computing, a number of performance metrics have been used over the years. The most common being *elapsed time*, *speedup*, and *efficiency*. The *elapsed time* (parallel runtime), T_P, is the time elapsed from the start of parallel computation to the moment when the last processor finishes execution. Beside workload W (problem size), it depends on the number of processors, P, the architecture of the parallel computing platform, and the algorithm. Therefore, the parallel runtime is measured by counting *computational* time and various classes of overheads: *unparallelised code*, *parallel start-up*, *synchronization*, *load imbalance*, and *communication* (routing) overheads (Sakellariou, 1996). The *speedup*, S, is defined as the ratio of the time taken to solve a problem on a single processor, T_S, using the best known version of a program to the time required to solve the same problem on a set of P parallel processors T_S/T_P. Finally, efficiency is defined as the ratio of of speedup to the number of processors S/P.

In order to evaluate the performance gains of the partitioning approach introduced in this work, several experi-

ments have been conducted. Our goal has been to justify and assess the effectiveness of the load balance achieved by the proposed approach, and to justify the theoretical results.

In essence, three mapping techniques have been evaluated; namely Cyclic Partitioning CP, Block Partitioning, and our ANOP technique. The benchmark used in our experiments contains a loop nest shown in Figure 1(b) (for $N = 300000$). These experiments were implemented using the *Java Parallel Virtual Machine* (JPVM) (Ferrari, 1997), which is an explicit message–passing based parallel programming interface library and similar to the well known (PVM) library. Furthermore, the experiments were carried out on a platform that consists of 8 personal computers (PC's). These computers were of $3000\ Mhz$ Pentium 4 processors with $512\ MB$ of DDR2-RAM and each runs its own *Windows XP* operating system. The nodes were interconnected using $100Mbps$ Ethernet local area network (LAN).

The experiments were run several times on different number of computers. Figure 2 shows the average total parallel computational time (elapsed time) of the loop nest (for $N = 300000$) on single, 2, 4, 6, and 8 computers. The figure depicts the performance of the three tested mapping approaches. It is clear that the proposed approach ANOP performs better than the BP and CP approaches.

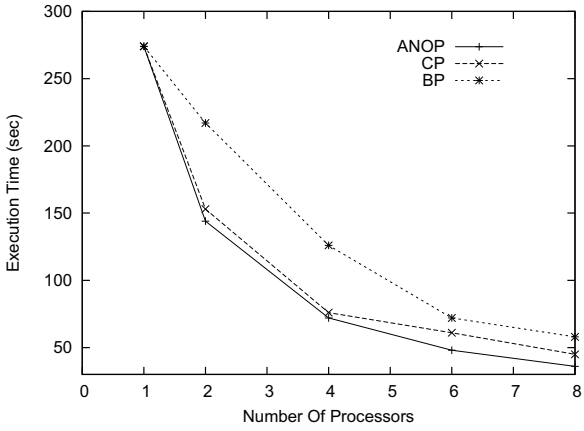


Figure 2: Execution Time by Different Partitioning Approaches and Number of Processors

Furthermore, Figure 3 reveals the *speedup* obtained by increasing number of processors by different approaches. It is clear that the ANOP achieves linear speedup and performs better than the other techniques. Though there is no interprocess communication and synchronization during runtime of the used benchmark program, the speedup is significantly lower than the number of processors, which is obvious. This can be related to the communication overhead caused by the used network platform. On the other hand, these figures show the impact of the load imbalance overhead, when using the BP and the CP techniques, on the elapsed time and speedup.

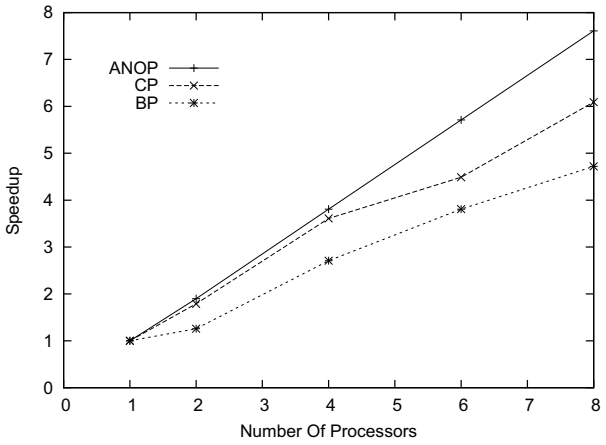


Figure 3: Speedup by Different Partitioning Approaches and Number of Processors

CONCLUSION AND FUTURE WORK

In this paper, we introduced a novel static simple intuitive approach for partitioning the iteration space of a perfect triangular loop nest (double loop nest) for homogenous environment of processors. We partition an iteration space along an axis corresponding to the outermost loop among processors and achieve a near–optimal partitioning. The partition thus obtained consists of contiguous and disjoint subsets, which facilitates exploitation of data locality. Moreover, the proposed approach can be easily implemented: unlike the other near optimal studied techniques, it does not need precomputation of the workload; a near optimal partitioning can be achieved by determining the lower and upper bounds (of the outermost loop index for each processor to carry out) as a function of the processor's number, total number of involved processors, and the upper bound of the original outermost loop.

The analytical results show that, using the proposed approach, we can achieve near optimal load balance and can minimize the load imbalance in parallel processing of a perfect triangular loop nest. Furthermore, the conducted experiments assess and justify the analytical results. The experimental results show the impact of the load imbalance on the elapsed time and the speedup achieved by different approaches.

As a future work, we would like to extend our approach to partition iteration space dynamically in a heterogenous environment.

REFERENCES

Beaumont, O.; V. Boudet; A. Legrand; F. Rastello; and Y. Robert. (2002). "Static Data Allocation and Load Balancing Techniques for Heterogeneous Systems". In *C.K. Yuen, editor, Annual Review of Scalable Computing*, volume 4, chapter 1. World Scientific.

Chaundhary, V. et al. (1996). "Design and Evaluation of an Environment APE for Automatic Parallelization of Programs".

In *Proceedings of the 1996 International Symposium on Parallel Architectures,* Algorithms and Networks Page: 77.

D'Hollander, E. H. (1992). "Partitioning and Labeling of Loops by Unimodular Transformations". *IEEE Transaction on Parallel and Distributed Systems,* 3(4):465-476.

Fahringer, T. (1998). "Efficient Symbolic Analysis for Parallelizing Compilers and Performance Estimator". *The journal of Super Computing*, 12, 227-252.

Ferrari, A. J. (1997). "JPVM:Network Parallel Computing in Java". Technical Report CS-97-29 Department of Computer Science University of Virginia, Charlottesville, VA 22903, USA. (December), available from: http://www.cs.virginia.edu/jpvm/doc/jpvm-97-29.pdf.

Haghighat, M. and C. Polychronopoulos. (1993). "Symbolic Analysis: A Basis for Parallelization, Optimization, and Scheduling of Programs". In *Proceedings of the Sixth Workshop Languages & Compilers for Parallel Computing*, (Aug.) 1993. available from: http://citeseer.ist.psu.edu/51605.html.

Haghighat, M. and C. Polychronopoulos. (1996). "Symbolic Analysis for Parallelizing Compilers". *ACM Transactions on Programming Languages and Systems*, 18(4):477-518,(July).

Hancock, D. J.; J. M. Bull; R. W. Ford; and T. Freeman. (2000). "Feedback Guided Dynamic Scheduling of Nested Loops". In *Proceedings of IEEE International Workshop on Parallel Processing* IEEE Computer Society Press, Pages 315-321, Elsevier Science (January), ISBN 0769507719.

Hancock, D. J.; J. M. Bull; R. W. Ford; and T. Freeman. (2000). "An Investigation of Feedback Guided Dynamic Scheduling of Nested Loops". In *Proceedings International Workshop on Parallel Processing 2000*, Toronto, Ont., Canada, pp 315-321. ISBN:0-7695-0771-9.

Hudak, D. E. and S. G. Abraham. (1990). "Compiler Techniques for Data Partitioning of Sequentially Iterated Parallel Loops". *ACM*.

Jialin, Ju. (1998). "Automatic Parallelization of Non-uniform Loops". PHD thesis, Jan, 1998 , Wayne State University, 155 pages; AAT 9915677, ISBN 9780599143999.

Kejariwal, A.; P. DAlberto; A. Nicolau; and C. D. Polychronopoulos. (2004). "A Geometric Approach for Partitioning N-Dimensional Non-Rectangular Iteration Spaces". In *Proceedings of the 17th International Workshop on Languages and Compilers for Parallel Computing*, West Lafayette.

Kejariwal, A.; A. Nicolau; U. Banerjee; C. Plychronopoulos. (2005). "A Novel Approach for Partitioning Iteration Spaces with Variable Densities". *Symposium on Principles and Practice of Parallel Programming*, (PPoPP),(June).

Kruskal, C. P. and A. Weiss. (1985). "Allocating Independent Subtasks on Parallel Processors". *IEEE Transactions on Software Engineering*, 11(10):10011016.

Lamport, L. (1974). "The Parallel Excecution of Do Loops". *Communication of the ACM*, 17-2,(Feb.), pp. 83-93.

Li, Y.; T. Callahan; E. Darnell; R. Harr; U. Kurkure; and J. Stockwood.(2000). "Hardware-software Co-design of Embedded Reconfigurable Architecture". Proceedings *37th conference on Design Automation*, pp 507-512, Los Angeles, Ca.

Petkov, D.; R. Harr; and S. Amarasinghe. (2002). "Efficient Pipelining of Nested Loops: Unroll-and-Squash". In *16th International Parallel and Distributed Processing Symposium*, Fort Lauderdale, Florida, (April).

Polychronopoulos, C.; D. J. Kuck; and D. A. Padua. (1986). "Execution of Parallel Loops on Parallel Processor Systems". In *Proceedings of the 1986 International Conference on Parallel processing*, page 519-527, (August).

Sakellariou, R. (1996). "On the Quest for Perfect Load Balance in Loop-Based Parallel Computations". PhD Thesis, Department of Computer Science, University of Manchester,1996. available at http://citeseer.ist.psu.edu/sakellariou98quest.html.

Scott, L. R.; T. Clark; and B. Bagheri. (2005). *Scientific Parallel Computing.* Princeton University Press.

Xue, C.; Z. Shao; M. Liu; and E. H.-M Sha. (2005). "Iterational Retiming: Maximize Iteration-Level Parallelism for Nested Loops." *Proceedings of the 2005 ACM/IEEE/IFIP International Conference on Hardware - Software Codesign and System Synthesis* (ISSS-CODES'05), New York, (Sept.).

AUTHOR BIOGRAPHIES

NEDAL M.S. KAFRI was born in Attil, Palestine and received his education at the Technical University of Plzeň in Czech Republic, where he obtained his MSc. degree in Control Systems and Installation Management in 1982. He worked at Al-Quds University as a lecturer at the Computer Science Department for fifteen years. Then he moved to the Czech Technical University in Prague in 1997, where he obtained his Ph.D. degree in Distributed Systems in 2002. Now he is a member of the academic staff of the Department of Computer Science of Al-Quds University in Palestine. His research interest is in Distributed and Parallel Computing. His email is `nkafri@science.alquds.edu` and his Webpage is `http://www.alquds.edu/staff/kafri`.

JAWAD ABU SBEIH was born in Hebron-Palestine in 1969 and obtained his first degree in Computer Science from Al-Quds Open University (QOU). His MSc degree was obtained from Al-Quds University in 2008. He is interested in parallel and distributed systems. He worked at Al-Quds Open University since 1993 as Teacher Assistant and the Continuous Learning Center coordinator at QOU. Now he is a lecturer in the department of computer science at QOU. His email address is `jabusbeih@qou.edu`. The website of QOU is `http://www.qou.edu`.