Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing (Minneapolis, March 1997).

Compile-Time Partitioning of Three-Dimensional Iteration Spaces

Rizos Sakellariou*

Abstract

This paper presents a strategy for compile-time partitioning of generalised three-dimensional iteration spaces; it can be applied to loop nests comprising two inner nested loops both of which have bounds linearly dependent on the index of the outermost parallel loop. The strategy is analysed using symbolic analysis techniques for enumerating loop iterations which can provide estimates for the load imbalance, and experimentally evaluated on a virtual shared memory parallel computer.

1 Introduction

Loop partitioning refers to this stage of the parallelisation process which deals with the formation of groups of loop iterations that can be executed in parallel; using a scheduling scheme, these groups are assigned to processors. Partitioning and scheduling constitute a fundamental problem to be solved on parallel computers [13]; this is generally termed as *mapping*. When mapping loop nests, it is essential to minimise overheads, such as load imbalance, communication, etc., thus increasing performance. Traditionally, researchers have been inclined towards run-time mapping schemes, on the basis that information not available at compile-time may permit a more balanced distribution of the workload [4], [9], [10], [15]. However, the latter may be achieved at the expense of additional overheads; furthermore, in the context of parallelising compilers, where a number of decisions are taken at compile-time, postponing the mapping phase until run-time may seriously affect the applicability of program restructuring transformations.

In this paper we describe a scheme for compile-time partitioning of loop nests comprising two inner nested loops both of which have bounds linearly dependent on the index of the outermost parallel loop. The main target is the minimisation of load imbalance, however, an attempt is also made to avoid options that may increase other sources of overhead. Load imbalance is estimated using symbolic analysis techniques for enumerating loop iterations; they are briefly introduced in section 2. Based on these, section 3 provides a description of the partitioning scheme, while preliminary experimental results on a virtual shared memory parallel computer demonstrate the efficiency of the method over other compile-time schemes.

2 Background

The loop nests considered in this paper have the form shown in figure 1. It is assumed that the sets of statements labelled `statements.1`, `statements.2`, ..., `statements.5` do

*Department of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, U.K.; email: rizos@cs.man.ac.uk

```
DOALL i = l1, u1
      (statements.1)
      DO j2 = l21 i + l22, u21 i + u22
         (statements.2)
         DO j3 = l31 i + l32 j2 + l33, u31 i + u32 j2 + u33
            (statements.3)
         ENDDO
         (statements.4)
      ENDDO
      (statements.5)
ENDDO
```

FIG. 1. *A canonical loop nest of depth* 3.

not contain statements whose execution depends on the value of the index of a surrounding loop; hence, the workload corresponding to each set of statements remains the same for any iteration of the outermost loop [1]. It is also assumed that the third set (`statements.3`) contains at least one statement, i.e. it is not empty, while the remaining sets of statements may be empty. The outermost loop, denoted by `DOALL` in the figure, is parallel, that is, its iterations can run concurrently on different processors.

Assuming that W_{tot} is the total amount of computation in the loop nest which is distributed amongst p processors in such a way that each processor i, $0 \leq i < p$, is assigned an amount of computation equal to W_i (clearly, $W_{tot} = \sum_{i=0}^{p-1} W_i$), then we say that this distribution exhibits a *load imbalance*, L, equal to

$$L = \max_i \left(W_i - \frac{W_{tot}}{p} \right) = W_{max} - \frac{W_{tot}}{p}, \tag{1}$$

where W_{max} is equal to $\max(W_0, W_1, \ldots, W_{p-1})$. The value of L is bounded between 0 and $W_{tot}(1 - 1/p)$. In the former case, that is, when, for all i, $W_i = W_{tot}/p$, we say that there exists a *perfect load balance*. Thus, reducing the overhead due to load imbalance is equivalent to finding $W_0, W_1, \ldots, W_{p-1}$ such that L in (1) is minimised.

In order to compute the values of $W_{tot}, W_i, 0 \leq i < p$, we consider the computational work corresponding to each set of statements of the loop body. Assume that W_I is the work corresponding to the statements which are executed only by the outermost loop (i.e., `statements.1` and `statements.5`), W_{J_2} is the work corresponding to the statements which are executed by the loop with index j_2 but not by the loop with index j_3 (i.e., `statements.2` and `statements.4`), and W_{J_3} is the work corresponding to the statements which are in the body of the loop with index j_3 (i.e., `statements.3`); then, the total amount of work in the loop nest, W_{tot}, is given by

$$W_{tot} = \sum_{i=l_1}^{u_1} W_I \; + \sum_{i=l_1}^{u_1} \sum_{j_2=l_{21}i+l_{22}}^{u_{21}i+u_{22}} W_{J_2} \; + \sum_{i=l_1}^{u_1} \sum_{j_2=l_{21}i+l_{22}}^{u_{21}i+u_{22}} \sum_{j_3=l_{31}i+l_{32}j_2+l_{33}}^{u_{31}i+u_{32}j_2+u_{33}} W_{J_3}.$$

The evaluation of sums such as the above depends on the number of symbolic variables involved in the loop bounds; detailed methodologies for the symbolic evaluation of sums in the context of parallelising compilers are described in [3], [11], [12], [14].

[1] This implies that the j_2 and/or j_3 loops may be surrounded by `DO` ... `ENDDO` loops which perform the same number of iterations regardless of the value of i; such loops may also exist in any of the five mentioned sets of statements.

3 Methodology

A subclass of the loop nests shown in figure 1 is defined as follows:

DEFINITION 3.1. *Consider the loop nest shown in figure 1; this is a* **canonical** *loop nest of depth 3, if and only if,* $u_1 > l_1$ *and, for all* i, j_2*, the following inequalities always hold*

$$l_{21}i + l_{22} \leq u_{21}i + u_{22}$$
$$l_{31}i + l_{32}j_2 + l_{33} \leq u_{31}i + u_{32}j_2 + u_{33}$$

where, $l_{21} \neq u_{21}$ *and at least one of the differences* $(l_{31} - u_{31})$, $(l_{32} - u_{32})$ *is non-zero* [2].

Definition 3.1 provides the basis for the following theorem.

THEOREM 3.1. *Consider a canonical loop nest of depth 3 as shown in figure 1; if the index of the outermost loop can be partitioned into* $2p^2$ *equal partitions, then the loop nest can be partitioned into* p *partitions of equal workload.*

Proof. Let $n = u_1 - l_1 + 1$ be the number of iterations of the outer loop; since the outer loop can be partitioned into $2p^2$ equal partitions, then $2p^2$ divides n. Thus, the k-th partition of the outermost loop will perform work W_k, equal to

$$W_k = \frac{n}{2p^2} W_I + I_{J_2} W_{J_2} + I_{J_3} W_{J_3}, \tag{2}$$

where I_{J_2}, I_{J_3} are the number of times the statements corresponding to work W_{J_2}, W_{J_3}, respectively, are executed by the k-th partition. Proceeding to the computation of a closed formula for I_{J_2}, I_{J_3}, we have to evaluate the summations

$$\begin{aligned} I_{J_2} &= \sum_{i=l_k}^{u_k} \sum_{j_2=l_{21}i+l_{22}}^{u_{21}i+u_{22}} 1, \\ I_{J_3} &= \sum_{i=l_k}^{u_k} \sum_{j_2=l_{21}i+l_{22}}^{u_{21}i+u_{22}} \sum_{j_3=l_{31}i+l_{32}j_2+l_{33}}^{u_{31}i+u_{32}j_2+u_{33}} 1, \end{aligned}$$

where $l_k = l_1 + kn/2p^2$ and $u_k = l_1 + (k+1)n/2p^2 - 1$. Since the loop nest is canonical by hypothesis, the upper bound of any of these sums is always greater than or equal to the corresponding lower bound. Thus, evaluating the sums, we get

$$\begin{aligned} I_{J_2} &= A_{12}k + A_{02}, \\ I_{J_3} &= A_{23}k^2 + A_{13}k + A_{03}, \end{aligned}$$

for A_{ij} constants. Therefore, equation (2) can be rewritten as

$$\begin{aligned} W_k &= \frac{n}{2p^2} W_I + (A_{12}k + A_{02})\, W_{J_2} + \left(A_{23}k^2 + A_{13}k + A_{03}\right) W_{J_3} \\ &= \frac{n}{2p^2} W_I + A_{02}W_{J_2} + A_{03}W_{J_3} + (A_{12}W_{J_2} + A_{13}W_{J_3})\, k + A_{23}W_{J_3}k^2 \\ &= C_0 + C_1 k + C_2 k^2, \end{aligned}$$

where C_0, C_1, C_2 are constants.

[2] This restriction guarantees that the number of iterations of each inner loop depends on the value of the index of at least one of the surrounding loops.

We assume that there is a way of grouping the $2p^2$ partitions along the index of the outermost loop into p partitions of equal workload; we call the latter *partitions of the loop nest* (as opposed to *partitions along the index of the outermost loop*). We further assume that each partition k', $0 \leq k' \leq p-1$, of the loop nest consists of $2p^2/p = 2p$ partitions along the index of the outermost loop and performs work $W'_{k'}$; then, for all k', it must be the case that

$$W'_0 = W'_1 = W'_2 = \ldots = W'_{p-1}.$$

Let $S_{k'} = \{s_{k'1}, s_{k'2}, \ldots, s_{k'2p}\}$ be the set of partitions along the index of the outermost loop which compose the k'-th partition of the loop nest; clearly, the integers s_{ij}, $0 \leq i < p$, $1 \leq j \leq 2p$ (that is, the elements of all the sets $S_{k'}$, $0 \leq k' < p$), are a permutation of the integers $0, 1, 2, \ldots, 2p^2 - 1$. Then, the workload, $W'_{k'}$, of the k'-th partition of the loop nest is given by

$$W'_{k'} = W_{s_{k'1}} + W_{s_{k'2}} + \ldots + W_{s_{k'2p}} = \sum_{i=1}^{2p} W_{s_{k'i}} = \sum_{i=1}^{2p} \left(C_0 + C_1 s_{k'i} + C_2 s_{k'i}^2 \right).$$

Since, for any two distinct partitions x, y of the loop nest, $0 \leq x, y < p$ and $x \neq y$, $W'_x = W'_y \iff W'_x - W'_y = 0$, we have

$$\begin{aligned} W'_x - W'_y &= \sum_{i=1}^{2p} \left(C_0 + C_1 s_{xi} + C_2 s_{xi}^2 \right) - \sum_{i=1}^{2p} \left(C_0 + C_1 s_{yi} + C_2 s_{yi}^2 \right) \\ &= \sum_{i=1}^{2p} \left(C_1 (s_{xi} - s_{yi}) + C_2 (s_{xi}^2 - s_{yi}^2) \right) \\ &= \left(\sum_{i=1}^{2p} s_{xi} - \sum_{i=1}^{2p} s_{yi} \right) C_1 + \left(\sum_{i=1}^{2p} s_{xi}^2 - \sum_{i=1}^{2p} s_{yi}^2 \right) C_2. \end{aligned}$$

The above expression must be equal to zero. A solution is given when the coefficients of C_1, C_2 are both equal to zero. In this case, the problem reduces to solving the following system of equations:

$$\begin{cases} \sum_{i=1}^{2p} s_{xi} = \sum_{i=1}^{2p} s_{yi} \\ \sum_{i=1}^{2p} s_{xi}^2 = \sum_{i=1}^{2p} s_{yi}^2. \end{cases}$$

Generalising the above for all partitions of the loop nest, we get the system

$$\begin{cases} \sum_{i=1}^{2p} s_{0i} = \sum_{i=1}^{2p} s_{1i} = \sum_{i=1}^{2p} s_{2i} = \ldots = \sum_{i=1}^{2p} s_{p-1,i} \\ \sum_{i=1}^{2p} s_{0i}^2 = \sum_{i=1}^{2p} s_{1i}^2 = \sum_{i=1}^{2p} s_{2i}^2 = \ldots = \sum_{i=1}^{2p} s_{p-1,i}^2. \end{cases} \tag{3}$$

Thus, all sets $S_{k'}$, $0 \leq k' < p$, must have equal sums of their elements and equal sums of the squares of their elements. Hence, if we find a method of partitioning all the integers from 0 to $2p^2 - 1$ into the p sets $S_0, S_1, \ldots, S_{p-1}$ so that (3) holds, the Theorem has been proved. We consider the p pairs given by

$$\{2pi + (k' + i) \bmod p, 2p(i+1) - 1 - (k' + i) \bmod p\},$$

```
DOALL I=1,N
   DO J=3,2*I+1
      DO K=J+2,5*I
         (statements)
      ENDDO
   ENDDO
ENDDO
```

FIG. 2. *An example of a canonical loop nest of depth 3.*

where $0 \leq i < p$. We observe that the sum of the elements of each pair is independent of k'. Furthermore, the sum of the squares of the elements of all the pairs is constant; this is because, for all k', $0 \leq k' < p$, the value of

$$((k'+0) \bmod p)^2 + ((k'+1) \bmod p)^2 + ((k'+2) \bmod p)^2 + \ldots + ((k'+p-1) \bmod p)^2$$

is equal to

$$0^2 + 1^2 + 2^2 + \ldots + (p-1)^2.$$

Given also that, for all i, k', any two such pairs have different elements, we have found a way of partitioning the integers $0, 1, 2, \ldots, 2p^2 - 1$ such that it yields a solution to (3). Hence, the set of partitions along the index of the outer loop which compose the k'-th partition of the loop nest is given by

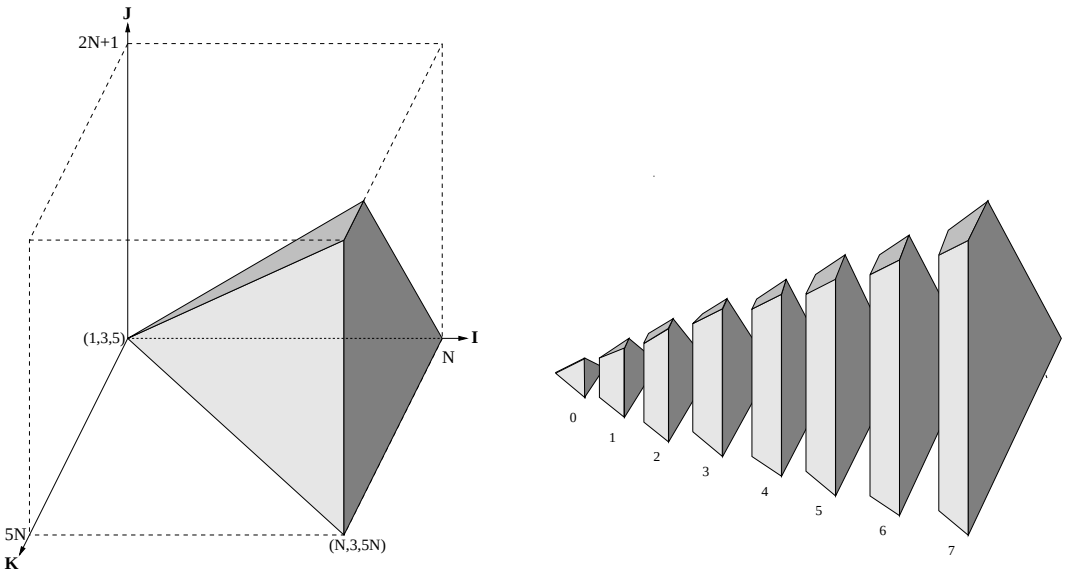
$$\begin{aligned} S_{k'} \quad = \quad & \{2pi + (k'+i) \bmod p : 0 \leq i < p\} \cup \\ & \{2p(i+1) - 1 - (k'+i) \bmod p : 0 \leq i < p\}\,. \end{aligned} \tag{4}$$

Therefore, the loop nest can be partitioned into p partitions of equal workload. □

In order to illustrate the results of Theorem 3.1, consider the loop nest shown in figure 2. Assuming that `N` > 1, then the inequalities `3` $\leq$ `2*I+1` and `J+2` $\leq$ `5*I` always hold, while, for each inequality, the coefficients of `I` are non-zero; hence, the requirements of Definition 3.1 are satisfied and the loop nest is a canonical loop nest of depth 3. Based on Theorem 3.1 and assuming that the number of iterations of the outer loop, `N`, is a multiple of $2p^2$, where p is the number of processors, partitioning the loop nest according to (4) leads to perfect load balance.

To provide an intuitive view of this partitioning scheme, the corresponding geometrical representation of the original loop nest is shown in figure 3.a. Assuming that two processors are used, the index of the outer loop is partitioned into $2 \cdot 2^2 = 8$ partitions; the corresponding polytope for each partition is shown in figure 3.b. Then, applying (4), the first processor is assigned partitions 0, 3, 5, and 6, while the second processor is assigned partitions 1, 2, 4, and 7; in geometrical terms, the polytopes have been assigned to two groups in such a way that the total volume of the polytopes in each group is the same.

The partitioning technique suggested in the proof of Theorem 3.1 can also be applied when the number of iterations of the outer loop, n, is not a multiple of $2p^2$; in this case, assuming that partitioning along the index of the outermost loop is performed as evenly as possible (that is, each partition has a number of iterations which differs by at most 1 from that of any other partition), a small value of load imbalance is expected. Theorem 3.1 can also be extended to cover cases where there are more than one inner loops at the same level (i.e., loops which are surrounded only by the same outer loops) whose bounds



a) Unpartitioned loop nest. *b) Partitioning into 8 partitions.*

FIG. 3. *Geometrical representation of the loop nest shown in* FIG. 2.

```
DO J=1,N
   DO I=1,J
      DO K=I,J
         A(I,J)=A(I,J)+B(I,K)*C(K,J)
      ENDDO
   ENDDO
ENDDO
```

FIG. 4. *Upper Triangular Matrix Multiplication.*

depend on the index of a surrounding loop; the necessary requirement is that, for any loop, the lower bound is always less than or equal to the upper bound. Finally, in the general case, where not all the inequalities in Definition 3.1 hold, index set splitting can be applied to transform the original loop nest into multiple adjacent loop nests, each of which satisfies the requirements of Definition 3.1 [12].

4 Experimental Results

In order to evaluate the performance gains of the partitioning strategy described in the previous section, we consider the parallelisation of the code shown in figure 4, which corresponds to the multiplication of two upper triangular $n \times n$ matrices [6] (the two outer loops have been interchanged in order to increase the number of unit stride array references). Clearly, the loop nest is canonical (see Definition 3.1), and, given that it has a depth of 3, a partitioning scheme based on the proof described in Theorem 3.1 may lead

TABLE 1

Execution time (in seconds) of upper triangular matrix multiplication for various mapping schemes on the KSR1.

N	Mapping scheme	Number of processors					
		1	2	4	8	12	16
256	KAP	4.03	3.64	2.78	1.71	1.12	0.97
	MARS	3.99	3.63	2.76	1.68	1.06	0.95
	CYC	4.02	2.05	1.07	0.55	0.40	0.35
	CAN	3.99	2.03	1.04	0.51	0.38	0.29
1024	KAP	527	458	308	174	128	110
	MARS	524	457	310	179	130	113
	CYC	525	267	154	69	55	49
	CAN	524	265	152	66	46	36

to perfect load balance. We compared this scheme (henceforth denoted by CAN) with three other compile-time mapping schemes, subsequently denoted by the shorthands KAP, MARS, and CYC; KAP corresponds to the mapping strategy of the KAP auto-parallelising compiler (which is based on scheduling chunks of consecutive iterations having a fixed size), MARS corresponds to the mapping strategy of the MARS experimental parallelising compiler [2] (which is equivalent to compile-time partitioning into a number of chunks of consecutive iterations equal to the number of processors), and CYC corresponds to a cyclic (or wrap [5]) way of mapping the iterations onto processors (i.e., processor 0 executes iterations $1, p+1, 2p+1, \ldots$, processor 1 executes iterations $2, p+2, 2p+2, \ldots$, in general, processor i, $0 \leq i \leq p-1$, executes iterations $i+1+kp, k = 0, 1, 2, \ldots, n/p-1$).

The parallelised programs were run on a KSR1, using two different values for N, 256 and 1024; their execution time is shown in Table 1. In both cases, KAP and MARS perform worst of all while CAN performs best. The performance of CYC is comparable with that of CAN when using a relatively small number of processors; for more than 12 processors, CYC causes a large number of cache misses.

5 Conclusion

The partitioning strategy described in this paper provides a mechanism for balancing the load of three-dimensional iteration spaces. Comparing to cyclic schemes of partitioning [5], the advantage of the presented scheme is that it is based on forming groups of consecutive iterations, an approach which is likely to reduce false sharing effects on certain machines [8], [9]. Furthermore, the presented scheme partitions the iterations of the outermost loop into equal (or almost equal) parts (as opposed to approaches analogous to balanced chunk scheduling [7]), which may be preferable for exploiting data parallelism at the program level.

References

[1] U. Banerjee, *Loop Transformations for Restructuring Compilers: The Foundations*, Kluwer Academic Publishers, 1993.

[2] F. Bodin, M. O'Boyle, *A Compiler Strategy for Shared Virtual Memories*, in B. K. Szymanski, B. Sinharoy (Eds.), Languages, Compilers and Run-Time Systems for Scalable Computers, Kluwer Academic Publishers, 1996, pp. 57–69.

[3] P. Clauss, *Counting Solutions to Linear and Nonlinear Constraints through Ehrhart polynomials: Applications to Analyze and Transform Scientific Programs*, Proceedings of the 1996 International Conference on Supercomputing (Philadelphia, May 1996), ACM Press, pp. 278–285.

[4] S. Flynn Hummel, E. Schonberg, L. E. Flynn, *Factoring: A Method for Scheduling Parallel Loops*, Communications of the ACM, 35 (8), Aug. 1992, pp. 90–101.

[5] A. Gerasoulis, I. Nelken, *Scheduling Linear Algebra Parallel Algorithms on MIMD Architectures*, in J. Dongarra, P. Messina, D. C. Sorensen, R. G. Voigt (Eds.), Proceedings of the 4th SIAM Conference on Parallel Processing for Scientific Computing, SIAM, 1989, pp. 68–95.

[6] G. H. Golub, C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, 1989.

[7] M. R. Haghighat, C. D. Polychronopoulos, *Symbolic Analysis for Parallelizing Compilers*, ACM Transactions on Programming Languages and Systems, 18 (4), July 1996, pp. 477–518.

[8] W. Li, *Compiler Optimizations for Cache Locality and Coherence*, Technical Report 504, Department of Computer Science, University of Rochester, Apr. 1994.

[9] D. J. Lilja, *Exploiting the Parallelism Available in Loops*, Computer, 27 (2), Feb. 1994, pp. 13–26.

[10] C. D. Polychronopoulos, D. J. Kuck, *Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers*, IEEE Transactions on Computers, 36 (12), Dec. 1987, pp. 1425–1439.

[11] W. Pugh, *Counting Solutions to Presburger Formulas: How and Why*, Proceedings of the ACM SIGPLAN '94 Conference on Programming Language Design and Implementation (Orlando, June 1994), ACM SIGPLAN Notices, 29 (6), June 1994, pp. 121–134; also available as Technical Report CS-TR-3234, Department of Computer Science, University of Maryland, Mar. 1994.

[12] R. Sakellariou, *On the Quest for Perfect Load Balance in Loop-Based Parallel Computations*, PhD Thesis, Department of Computer Science, University of Manchester, U.K., 1996.

[13] V. Sarkar, *Partitioning and Scheduling Parallel Programs for Multiprocessors*, Research Monographs in Parallel and Distributed Computing Series, MIT Press, 1989.

[14] N. Tawbi, *Estimation of Nested Loops execution time by Integer Arithmetic in Convex Polyhedra*, Proceedings of the 8th International Parallel Processing Symposium, IEEE Computer Society Press, 1994, pp. 217–221.

[15] T. H. Tzen, L. M. Ni, *Trapezoid Self-Scheduling: A Practical Scheduling Scheme for Parallel Computers*, IEEE Transactions on Parallel and Distributed Systems, 4 (1), Jan. 1993, pp. 87–98.