J. King Saud Univ., Vol. 11, *Comp. & Info. Sci.*, pp. 1-23 (A.H. 1419/1999)

A Vector-space Model for Parallel Workload Characterization

Abdullah I. Al-Meajil[*], Tarek El-Ghazawi[*] and Thomas Sterlings[**]
[*]*Department of Electrical Engineering and Computer Science,*
George Washington University, USA
[**]*Center of Excellence in Space Data & Information Sciences (CESDIS)*
NASA/Goddard Space Flight Center, USA

(Received 27 December 1995; accepted for publication 26 November 1996)

Abstract. Software parallelism is a key factor in performance of parallel systems. In this paper we discuss a parallel-instruction vector space model for workload representation and comparison. This model will be compared with the parallelism-matrix technique, which is based on the Frobenius matrix norm. The latter compares two workloads based on identical parallel instructions only, whereas the former compares two workloads based on all parallel instructions. It will be shown that the parallel-instruction vector space method outperforms the parallelism-matrix method in time and space, as well as in accuracy. Further, it will be shown that this model provides a useful framework for the design and analysis of benchmarks. This will be demonstrated by analyzing some of the NASA/NAS Parallel Benchmark workloads and their performance measurements in the context of the model.

Keywords: Instruction-level Parallelism; Benchmarking; Workload Characterization; Performance Evaluation.

1. Introduction

Many efforts have tackled the problem of characterizing and measuring specific aspects of parallel workloads. Depending on the purpose of the work, these effects have quantified attributes such as the total number of operations, average degree of parallelism, and instruction mixes [1-15]. More work, however, is still needed in order to characterize parallel workloads based on how they are expected to exercise parallel architecture. Such characterization has to be valid across a wide range of parallel architectures.

This work has been partially supported by NASA High-Performance Computing and Communications (HPCC) program through CESDIS/USRA.

Therefore, we propose an architecture-invariant characterization which uses an abstract parallel machine to provide understanding of essential workload aspects that can impact performance and requirements [16]. This characterization takes into account the type of operations and operation counts presented to the machine on a cycle-by-cycle basis, as given by the dynamic parallel instruction sequence in workloads.

Since measuring parallel instructions is of interest to this study, we consider efforts that examined instruction-level parallelism. Researchers have measured instruction-level parallelism to try different parallel compilation concepts and study their effect on parallelism. Most of these studies measured the limits of (average) parallelism under ideal conditions, such as the oracle model where parallelism is only limited by true flow dependencies. Then, they examined the drop in parallelism when specific architectural or compilation implementation concepts were introduced into the model.

Studies on instruction-level parallelism have taken one of two approaches. One approach is to analyze the selected workload statically at the source-code level (or object-code with a special interpreter based on a certain machine) [1-3]. The other approach is to collect dynamic traces from actual execution and schedule the instructions on the target machine model [4-12]. The static analysis of workloads tends to give conservative estimates for available parallelism since control dependencies can be only resolved at run time. On the other hand, the dynamic analysis of workloads using speculative execution and branch prediction [17] can measure the amount of parallelism which theoretically exists in a given workload. Although the scope was different in these studies, the techniques are of interest to our work as alternative means of measuring parallelism. Many researchers have observed that benchmarking should become more of a scientific activity [18]. Due to the necessity of parallelism for achieving good performance, this work develops a well founded parallelism-based workload representation and comparison framework [19]. The framework provides meaningful information to designers and users of high-performance systems as well as to parallel benchmarking developers and analysts.

In the present work we only consider workload characterization based on parallel instructions, which encompasses information on parallelism, instruction mix, and amount and type of work on a cycle-by-cycle basis. Bradley and Larson [20] have considered parallel workload characterization using parallel instructions. Their technique compares the differences between workloads based on executed parallel instructions. Executed parallelism is the parallelism exploited as a result of interaction between hardware and software. This technique is, therefore, an architecture-dependent technique due to its dependency on the specific details of the underlying architecture. In their study, a subset of the Perfect Benchmarks has been chosen to run on the Cray Y-MP. Then a multidimensional matrix that represents the workload parallelism profile was constructed. The Frobenius matrix norm is then used to quantify the difference between

the two workload parallelism matrices. In addition to requiring a lot of space and time, this method is restricted to comparing identical executed parallel instructions only. On the other hand, the technique proposed here uses the vector-space model to represent parallel workloads and measure the degree of similarity between them. In this workload model, each parallel instruction is represented as a point in a multidimensional space, where each dimension represents an operation/instruction type. Each workload in a benchmark suite can be then approximated by a parallel-instruction centroid. Thus, the difference between two workloads can be quantified using appropriately normalized Euclidean distance between the two centroids.

Architecture-invariant of our parallel-instruction vector-space model is derived from using the oracle abstract architecture model [4,12]. The oracle model is an idealistic model that considers only true flow dependencies. The parallel instructions (PIs) are generated by scheduling sequential instructions that are traced from a RISC processor execution onto the oracle model. The traced instructions are packed into parallel instructions while respecting all flow dependencies between instructions. To compare our technique with the parallelism-matrix one, we consider an extended version of the parallelism-matrix technique which is made architecture-invariant by replacing the Cray Y-MP simulator with the oracle model.

In this paper we present the concept of parallel-instruction vector space model and a parallel-instruction workload similarity measurement technique. We compare this technique to the parallelism-matrix method [20]. It will be shown here that our method is machine-invariant and better represents the degree of similarity between workloads. Further, the technique is very cost efficient when compared with similar methods. We also show that the parallel-instruction vector space model provides a useful framework for the design and analysis of benchmarks. This is demonstrated by analyzing some of the NAS Parallel Benchmark workloads [21, 22] and their performance measurements using this model. The NAS Parallel Benchmark (NPB) suite is rooted in the problems of computational fluid dynamics (CFD) and computational aerosciences. It consists of eight benchmark problems each of which is focusing on some important aspect of highly parallel supercomputing, for aerophysics applications [23, 24]. This paper is organized as follows. Section 2 presents an overview of previous work, while section 3 presents our parallel-instruction vector space model in details. The comparison between the two techniques is discussed in the section 4. Similarity comparisons for the NASA/NAS Parallel benchmarks will be presented in the section 5. Finally, conclusions and future directions of research are presented in section 5.

2. The Parallelism-Matrix Technique

This technique represents an executed-parallelism workload profile in a multidimensional matrix (referred to as a *n*-matrix). Each dimension in this *n*-matrix

represents a different instruction type in a workload. "Work" has been defined to be the total number of operations of interest a workload can have. When there is only one instruction type of interest, work is considered to be the total operations of that type in a workload. Therefore, a natural extension to the simple post-mortem average is a histogram $W = <W_0, \quad , W_f>$, where W_i is the number of clock periods during which i operations of interest type were completed simultaneously. The sum

$$t = \sum_{i=0}^{f} W_i \tag{1}$$

is the number of clock periods consumed by the entire workload, and the weighted sum

$$w = \sum_{i=0}^{f} iW_i \tag{2}$$

is the total amount of work performed by the workload. To facilitate comparisons between workloads that have different execution times, each entry in the histogram is divided by t, the total execution time in clock periods, to produce a normalized histogram called the *parallelism vector* $P = <P_0, \quad , P_f>$, where $P_i = W_i/t$. By construction, each entry P_i has a value between 0.0 and 1.0 that indicates the fraction of time during which i units of work were completed in parallel.

In a similar way, executed parallelism matrices of arbitrary dimension can be constructed with one dimension for each of the different kinds of work that are of interest. Some other possibilities for "work" include logical operations, integer operations, and I/O operations. Depending on how work is defined, various parallelism profiles from an executed-parallelism workload matrix can be obtained.

To illustrate a two-dimensional case, Table 1 shows a two-dimensional matrix that represents a parallelism profile for ARC2D workload which represents the Aerodynamics application area in the Perfect Benchmarks suite. This parallelism matrix has been the output of the CRAY Y-MP simulator that has three floating-point functional units (add, multiply, and reciprocal approximation) and three memory units (two load and one store). In this matrix the row index represents the multiplicity of memory operation in a parallel instruction, while columns represent the multiplicity of the floating-point operation. Each cell position, therefore, indicates a possible mix in a parallel instruction. The value in the cell, however, indicates the fraction of such instruction in the workload. Taking cell (3,0), e.g., it indicates that 1% of the parallel instruction contain 3 memory instructions but no floating-point operation. In this example, the vector of row sums of the parallelism matrix gives a profile of the memory

executed parallelism. On the other hand, the vector of column sums shows the profile of floating-point executed parallelism.

Table 1. Parallelism matrix for ARC2D in the perfect benchmark suite

		FP			
		0	**1**	**2**	**3**
M	3	0.01	0.03	0.03	0.00
E	2	0.05	0.13	0.08	0.00
M	1	0.07	0.20	0.11	0.00
	0	0.07	0.13	0.07	0.02

The parallelism profiles for two workloads, thus far, can be compared by comparing the parallelism matrices for each workload using the Frobenius matrix norm to quantify the difference. If A is the two-dimensional m n parallelism matrix for workload 1 and B is the m n parallelism matrix for workload 2, then the difference in executed parallelism between the two workloads can be gauged by

$$\text{Diff}(A, B) = \| A - B \|_F = \sqrt{\sum_{i=0}^{m} \sum_{j=0}^{n} | a_{ij} - b_{ij} |^2} \qquad (3)$$

Intuitively, the Frobenius norm represents the "distance" between two matrices, just as the Euclidean formula is used to measure the distance between two points. This distance may range from *0.00*, for two workloads with identical executed parallelism distributions, to $\sqrt{2}$ in the case where each matrix has only one non-zero element (with value *1.00*) in a different location.

3. The Parallel-Instruction Vector-Space Model

Our Parallel-Instruction Vector Space Model is represented here provides for an effective workload representation (characterization), as will be shown. Effectiveness, in this regard, refers to the fidelity of the representation and the associated space and time costs. In this framework, each parallel instruction can be represented by a vector in a multidimensional space, where each coordinate corresponds to a different instruction type (*I-type*) or a different basic operation (ADD, LOAD, FMUL,). The position of each parallel instruction in the space is determined by the magnitude of the *I-type*s in that vector.

Parallel workload instant and parallel work: The workload instant for a parallel computer system is defined here as the types and multiplicity of operations presented for

execution by an idealistic system (oracle model), in one cycle. A workload instant is, therefore, represented as a vector quantity (parallel instruction) where each dimension represents an operation type and the associated magnitude represents the multiplicity of that operation in the parallel instruction. Parallel workload of an application is the sequence of instances (parallel instructions) generated from that application.

Workload centroid: The centroid is a parallel instruction in which each component corresponds to the average occurrence of the corresponding operation type over all parallel instructions in the workload. Centroid, therefore, can be thought of as the point mass for the parallel workload body.

Workload similarity: Two workloads exhibited by two applications are, thus, considered identical if they present the machine with the same sequence of parallel instructions. In this case both workloads are said to be exercising the machine resources in the same fashion.

3.1 A vector-space model for workload

Consider three types of operations (*I-types*) such as arithmetic operations (INT), floating-point operations (FP), and memory access operations (MEM), then the parallel-instruction vector can be represented in a three-dimensional space as a triplet:

$$PI = (MEM, FP, INT).$$

If an instance of parallel instructions in a workload is given by

$$PI_i = (4, 7, 2),$$

then this ith parallel instruction in the workload has 4 MEM operations, 7 FP operations, and 2 INT operations. The total operations in this parallel instruction would be 13 operations that can be run simultaneously. In general, parallel instructions are represented as t-vectors of the form

$$PI_i = (a_{i1}, a_{i2}, \ldots, a_{it}) \tag{4}$$

where the coefficient a_{ik} represents the count of instructions of type k in parallel instruction PI_i.

Comparing workloads based on sequence of parallel instructions could be quite complex and prohibitive, for realistic workloads. This is because the comparison requires examining each parallel instruction from one workload against all parallel instructions in the other workload, which has very high computational and storage

requirements. This has led us to propose the concept of centroid for workload representation and comparison, which is a cost-effective means to represent workloads. The centroid is similar to the center of gravity of a set of masses, see Fig. 1. The centroid is a parallel instruction in which each component corresponds to the average occurrence of the corresponding instruction type over all parallel instructions in the workload. Given a set of n parallel instructions constituting a certain workload, the corresponding centroid vector is

$$C = (C_1, C_2, \quad , C_t) \tag{5}$$

where:

$$C_k = 1/n \sum_{i=1}^{n} a_{ik} \tag{6}$$

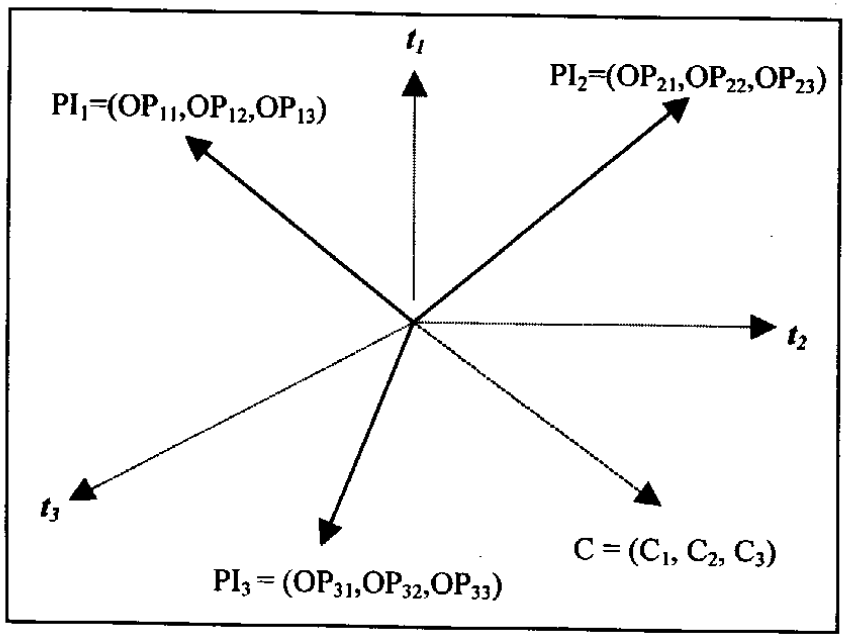

Fig.1. Vector representation of parallel instructions and their centroid in a 3D-space.

To illustrate the above, Table 2 shows the process to generate a centroid vector for a workload.

Table 2. Example of the workload representation

$OP_1, OP_2, OP_3, OP_4, \ldots, OP_{23}, OP_{24}$

(a): *Collect the Dynamic Trace for the Underlying Application.*

	$OpType_1$	$OpType_2$	$OpType_3$	$OpType_4$	$OpType_5$
PI_1	1	3	0	4	0
PI_2	0	2	0	3	1
PI_3	0	7	0	2	1

(b): *Schedule on Oracle to Generate Stream of PIs.*

	$OpType_1$	$OpType_2$	$OpType_3$	$OpType_4$	$OpType_5$
C	1/3	4	0	3	2/3

(c): *Obtain Centroid for PIs in a Workload.*

In addition to simplifying the analysis, centroids have the quality of providing an easy way to grasp the workload characteristics and the corresponding resource requirements. This is because the centroid couples instruction-level parallelism and instruction mix information to represent the types and multiplicity of operations that the machine is required to perform, on the average, in one cycle. This also represents the functional units types and average number of them needed in the target machine in order to sustain a performance rate close to the machine's peak rate, under such kind of workloads. Due to their simplicity and physical significance, as discussed above, centroids are used in the rest of this work as the basis for workloads representation and comparisons.

3.2 Workload comparison using the vector-space model

Measuring similarity based on centroids mandates the selection of a similarity metric which can generate easy to understand real-valued numbers. To do so, we propose the following metric characteristics:

- metric generates normalized values between 0 and 1;
- "0" represents one extreme (e.g. similar), while "1" represents the other extreme (e.g. dissimilar);
- scales appropriately between these two extremes as the similarity between the compared workloads changes.

This leads us to select the normalized Euclidean distance between two centroids, representing two different workloads, as follow. Let point u be the t-tuple $(a_1, a_2, \ldots,$

a_t) and point v be the t-tuple (b_1, b_2, , b_t); then the Euclidean distance, from Pythagoras' theorem, is

$$d(u,v) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_t - b_t)^2} \quad (7)$$

In order to conform with the aforementioned metric characteristics, distance between any two workloads in a benchmark suite can be normalized by dividing the distance between the two workloads by the maximum distance found in that two workloads, from the origin. Let WL_r and WL_s be two workloads in a benchmark suite, where each can be characterized by a t-centroid vector (t instruction types) as follows:

$$WL_r = (C_{r1}, C_{r2}, , C_{rt}) \text{ and } WL_s = (C_{s1}, C_{s2}, , C_{st}).$$

And let C_{ik} represent the centroid magnitude of the kth instruction type in workload i. The maximum centroid-vector in this workloads can be represented as follows.

$$C_{max}(WL_r, WL_s) = (\max(C_{r1},C_{s1}), \max(C_{r2},C_{s2}), \ldots, \max(C_{rt},C_{st})) \quad (8)$$

Then, the similarity between the two workloads can be measured as:

$$Sim(WL_r,WL_s)=d(WL_r,WL_s) / d(C_{max}, \text{null-vector}) \quad (9)$$

where null-vector is a t-vector in which each element equals to 0; hence, *null-vector* = *(0,0,...,0)*. In this case, 0 represents identical workloads while 1 represents orthogonal workloads that use different operations and thus, would exercise different aspects of the target machine.

4. Comparison Study for the Two Techniques

4.1 Examples

Sample examples have been developed in order to demonstrate how this method and our method compare. Let us have a benchmark suite that consists of five benchmark workloads. Table 3 shows the five sample workloads. Each workload is presented in a table of size i j, where i is the total number of unique parallel instructions in the workload and j has a length of t = 3 where each one of the t columns represents an operation type (Memory, Floating-Point, or Integer). The additional column, *PI-Num*, represents the total number of instances for that unique parallel instruction. For example,

when *PI-Num* equals 7 at the second row in the fourth sample workload, then it means that there are 7 instances of this unique parallel instruction PI = (3, 4, 2) in this fourth workload.

Table 3. A sample benchmark suite of five workloads

Workload-1

MEM	FP	INT	*PI-Num*
1	0	1	*5*
0	1	0	*3*
1	0	0	*7*
0	0	1	*2*

Workload-2

MEM	FP	INT	*PI-Num*
0	1	1	*2*
1	1	0	*3*
1	0	1	*7*
1	1	1	*5*

Workload-3

MEM	FP	INT	*PI-Num*
3	2	1	*5*
4	3	0	*7*
2	3	1	*2*
2	3	0	*3*

Workload-4

MEM	FP	INT	*PI-Num*
4	3	2	*3*
3	4	2	*7*
4	4	1	*2*
4	4	2	*5*

Workload-5

MEM	FP	INT	*PI-Num*
0	2	0	*3*
2	0	0	*7*
1	0	2	*5*
0	0	2	*2*

MEM: Memory operations; FP: Floating-point operations; INT: Integer operations
PI-Num: Number of instances of a unique parallel instruction

4.2 Parallelism-matrix measurements

In the parallelism-matrix technique, each workload parallelism profile is presented in a three-dimensional matrix. For example, workload WL_3 is illustrated in Table 4 by spreading the *INT*-dimension layers of the three-dimensional matrix over two layers for readability. Table 5a. represents the 1st INT layer where no INT operations are in the corresponding parallel instruction. Figure 5b represents the 2nd INT layer when only one INT operation is in the parallel instruction.

Table 4. Parallelism matrix representation for workload WL_3

4.a. For the 1st INT layer

		FP			
		0	**1**	**2**	**3**
	4	0.00	0.00	0.00	0.412
M	3	0.00	0.00	0.00	0.00
E	2	0.00	0.00	0.00	0.176
M	1	0.00	0.00	0.00	0.00
	0	0.00	0.00	0.00	0.00

4.b. Parallelism matrix for the 2nd INT layer

		FP			
		0	1	2	3
	4	0.00	0.00	0.00	0.00
M	3	0.00	0.00	0.294	0.00
E	2	0.00	0.00	0.00	0.118
M	1	0.00	0.00	0.00	0.00
	0	0.00	0.00	0.00	0.00

To compare two workloads, the Frobenius matrix norm is used in order to quantify the distances or differences. Recall that the parallelism-matrix technique has been extended to be architecture-invariant for comparisons with the parallel-instruction vector space model. As mentioned before, the Frobenius norm ranges between 0.00 and $\sqrt{2}$, therefore, it will be divided by that value. Table 5 presents similarity measurements for some pairs of workloads in the benchmarks set.

Table 5. Similarity measurements usingparallelism-matrix technique

	Parallelism-matrix
WL_1&WL_2	0.424
WL_1&WL_3	0.549
WL_1&WL_4	0.549
WL_1&WL_5	0.549
WL_3&WL_4	0.549

4.3 Parallel-instruction vector space measurements

In the aforementioned vector-space model, each workload centroid is calculated from all the parallel instructions that are in a workload. All workload centroids are presented in Table 6 where the row represents different workload and the column represents the instruction type in the workload.

Table 6. Workload centroids

	MEM	FP	INT
WL_1	0.706	0.177	0.412
WL_2	0.883	0.589	0.824
WL_3	3.12	2.71	0.412
WL_4	3.588	3.824	1.882
WL_5	1.118	0.353	0.824

Recall that the distance between two workload centroid points needs to be normalized to produce numbers in the range of *0.00* and *1.00*. Table 7 presents similarity measurements for some pairs of workloads in our benchmark suite when the parallel-instruction vector space model is used. Note that *1.00* means dissimilar and *0.00* means identical.

Table 7. Similarity measurements using parallel instruction vector space technique

	Parallel-instruction Vector Space
WL_1&WL_2	0.45318
WL_1&WL_3	0.8425
WL_1&WL_4	0.8751
WL_1&WL_5	0.1804
WL_3&WL_4	0.650

4.4 Discussion

The similarity among the workloads in the example suite can be examined quantitatively using similarity functions, expressions (3) and (9). Table 8 shows the quantitative similarity for some pairs of workloads when the two techniques are used. Note that similarity in parallelism is not a transitive relation.

Table 8. Workload similarity in the example benchmarks with the two techniques

	Parallelism-Matrix	Parallel-instruction Vector Space
WL_1&WL_2	0.424	0.45318
WL_1&WL_3	0.549	0.8425
WL_1&WL_4	0.549	0.8751
WL_1&WL_5	0.549	0.1804
WL_3&WL_4	0.549	0.650

Evidently, measurements obtained by parallelism-matrix technique have more shortcomings.For example, in the parallelism-matrix the similarity value of the workload comparisons WL_1 & WL_3, WL_1 & WL_4, and WL_1 & WL_5 are all 0.549. This value does not change in these cases because of the absentia of the identical parallel instructions from the workloads. However, if there are some identical parallel instructions in workloads, then the similarity value may have more meaningful values. For example, in comparing WL_1 & WL_2 the similarity value is 0.424. The reason is that, both workloads have a common identical parallel instruction. Hence, the parallelism-matrix technique

lacks the ability to compare realistic workloads when they lack identical parallel instructions. This is the case even when the parallel instructions of the two workloads are quite similar but not identical. In the parallel-instruction vector space technique, Table 4 shows more meaningful values. When two workloads are quite different, the similarity values are high as in the case of WL_1 & WL_4. On the other hand, when there are some differences in the workloads, the similarity value changes proportionally. For example, WL_1 and WL_2 behave almost in the same manner. By applying the parallel-instruction vector space technique the similarity between these two workloads equals 0.45318, while 0.549 is produced by the parallelism-matrix technique. A similar scenario occurs when WL_1 & WL_5 are compared.

In general, the parallel-instruction vector space method presents more detailed information. For each workload centroid, each attribute represents an arithmetic mean of a type of instruction in the workload. By comparing this centroid to other workload centroid, each matching attribute will be compared. This comparison tells in which direction these two workloads are different. Considering workloads WL_1 and WL_3, along the arithmetic instruction type, these two workloads exercise the oracle model in the same manner. However, at the floating-point instruction type, WL_3 uses more floating-point functional units than WL_1.

The parallel-instruction vector space method is also more efficient in time and space. After producing parallel instructions, both techniques make two steps in order to measure the workload similarity. The first step is workload representation, and the second is workload comparison. The parallelism-matrix technique represents a workload in a t-dimensional matrix where each dimension represents an instruction type. The maximum magnitude of a dimension is $n + 1$, where n represents the maximum instruction type occurrences in any parallel instruction in that workload. Therefore, the parallel matrix technique needs as much storage as the size of the matrix. This has storage complexity of $O(n^t)$. On the other hand, the parallel-instruction vector space model represents a workload by a centroid of length t. Therefore, the storage complexity of this technique is $O(t)$. The time for workload representation, in the parallelism-matrix technique, takes the parallel-instruction counts (p) times the parallel-instruction length (t), or $O(p \cdot t)$. This is because all parallel instructions have to be generated first, before constructing and filling the matrix. However, in the parallel-instruction vector space model, the computational complexity is $O(t)$. This is due to the fact that the workload centroid is calculated on-the-fly.

In the comparison step (measuring similarity), the parallelism-matrix technique compares every element of one matrix with the corresponding element in the other matrix. Therefore, the computational complexity of this technique is $O(n^t)$. In the

parallel-instruction vector space model, however, the computational complexity is $O(t)$. This is due to the fact that the workload centroid has t types of instructions.

Table 9 summarizes the comparative study between the parallelism-matrix technique and the parallel-instruction vector space technique. It shows that our parallel-instruction vector space model outperforms the parallelism-matrix technique for measuring the workload similarity in all essential aspects.

Table 9. Comparison parameters for both techniques

	Parallelism-matrix	Parallel-instruction vector space
Representation Cost (time)	$O(p \cdot t)$	O(t)
Representation Cost (storage)	$O(n^t)$	O(t)
Comparison-Cost	$O(n^t)$	O(t)
Accuracy	Depends only on identical PIs	Depends on all PIs
Machine-Dependency	Architecture-dependent†	Architecture-invariant

†original parallelism-matrix technique [14]
p: parallel-instruction count. **t:** parallel-instruction size. **n:** maximum dimension length

5. NAS Parallel Benchmark Workload Comparison

In order to demonstrate the utility of this model and verify the underlying concepts with real-life applications we consider to study the NASA Parallel Benchmark suite [23, 24] using our model. We start by representing the workloads in this suite as well as characterizing the similarity among different workload pairs. Then, we examine some of NASA reported performance measurements [21] to demonstrate how workload similarity, in the context of our model, could lead to similarity in performance.

5.1 A NAS parallel benchmark overview

The NPB suite consists of two major components: five parallel kernel benchmarks and three simulated computational fluid dynamics (CFD) application benchmarks. This benchmark suite successfully addresses many of the problems associated with benchmarking parallel machines. They intended to accurately represent the principal computational and data movement requirements of modern CFD applications. An exhaustive description of these NPB problems is given in [16, 17, 20].

embar *is to execute 2^{28} iterations of a loop in which a pair of random numbers generated and tested for whether Gaussian random deviates made from them according to a specific scheme. It is typical of many Monte Carlo applications: Two-dimensional statistics accumulated from a large number of Gaussian pseudorandom numbers, generated according to a scheme that well suited for parallel computation. This kernel is termed "embarrassingly parallel," based on the trivial partitionability of the problem,*

while incurring no data or functional dependencies. It is included in the NPB suite to establish the reference point for peak performance on a given platform.

mgrid *is to execute four iterations of the V-cycle multigrid algorithm to obtain an approximate solution to the discrete Poisson problem* ^{2} *= v on a 256 256 256 grid with periodic boundary conditions. The problem is simplified in that it has constant rather than variable coefficients, as in more realistic applications.*

cgm *is to use the power and conjugate gradient methods to approximate the smallest eigenvalue of a large, sparse, symmetric positive definite matrix of order 14,000 with a random pattern of nonzeros. This problem is typical of unstructured grid computations and it uses sparse matrix-vector multiplication.*

fftpde *uses FFT's on 256 256 128 complex array to solve a three-dimensional partial differential equation. This benchmark represents the essence of many "spectral" codes or eddies turbulence simulations.*

buk *is to perform 10 ranks of* 2^{23} *integer keys in the range* $[0, 2^{19}]$*. This kernel implements a sorting technique that is important in "particle method" codes. It is similar to "particle in cell" physics applications, where particles assigned to cells and may drift out. The sorting operation reassigns particles to the appropriate cells. This problem is unique in that floating-point arithmetic is not involved.*

applu *does not perform an LU factorization, but instead uses a symmetric, successive over relaxation numerical scheme to solve a regular-sparse, block (5 5) lower and upper triangular system. This problem represents the computations associated with the implicit operator of a newer class of implicit CFD algorithms, typified at NASA Ames by the code INS3D-LU. This problem exhibits a somewhat limited amount of parallelism compared to the next two simulated CFD applications. A complete solution of this benchmark requires 250 iterations.*

appsp *is a solution of multiple, independent systems of non-diagonally-dominant, scalar pentadiagonal equations. A complete solution requires 400 iterations.*

appbt *is a solution of multiple, independent systems of non-diagonally-dominant, block tridiagonal equations with a (5 5) block size. A complete solution requires 200 iterations. appsp and appbt are representative of computations associated with the implicit operators of CFD codes such as ARC3D at NASA Ames.*

In order to keep traces and analysis time within practical limits, we have used the short input files provided by the NAS Parallel Benchmark suite. The sample codes, provided by NAS, actually solve scaled-down versions of the benchmarks that run on many current-generation workstations. The standard input sizes for the NPB suites

referred to as the Class A and Class B size problems. Table 10 lists the problem size [17] and the dynamic operation counts of: the sample code problems, the Class A problems, and Class B problems. Operation counts are obtained using the spy tool [21].

Table 10. Operation counts for NAS "sample, class A, and class B" benchmarks running on one processor at NASA/GSFC

Benchmarks	Problem size			Dynamic operation count (10^9)		
	Sample	Class A	Class B	Sample	Class A	Class B
embar	2^{24}	2^{28}	2^{30}	0.3911	26.68	1008.8
mgrid	32^3	256^3	256^3 †	0.1154	3.905	18.81
cgm	10^5	14,000	75,000	0.0161	1.508	54.89
fftpde	64^3	256^2 128	256^2 512	1.5230	5.631	71.37
buk	2^{16}	2^{23} 2^{19}	2^{25} 2^{21}	0.0768	0.7812	3.150
applu	12^3	64^3	102^3	0.5088	64.57	319.6
appsp	12^3	64^3	102^3	0.8920	102.0	447.1
appbt	12^3	64^3	102^3	1.1157	181.3	721.5

† code is different from class a [17]

5.2 Analysis process

In order to explore the inherent parallelism in workloads, instructions traced are scheduled for the oracle model architecture. This model presents the most ideal machine that have unlimited processors and memory, and does not incur any overhead. The Sequential Instruction Trace Analyzer (SITA) is a tool developed at McGill University to measure the amount of parallelism which theoretically exists in a given workload [11, 22]. SITA takes a dynamic trace generated by spy tool from a sequential execution of a conventional program, and schedules the instructions according to how they could be executed on an idealized architecture while respecting all relevant dependencies between instructions. Currently, SITA is used to analyze SPARC executables and is designed to work with spy tool, which is the only tool needed from the Spa package [21]. SITA tool includes a pre-analyzer (*sitapa*), a control-dependence analyzer (*sitadep*), and a trace scheduler(*sitarun*). Note that traced processes have been observed to run about 40 times slower than normal. If spy is used with a trace analyzer, such as sitapa or sitarun, the resulting system will run some 400-600 times slower than normal (400 for oracle and 600 for other models).

The analysis process of a SPARC workload or benchmark takes four steps. First, a SPARC executable file is created, using the desired optimization level. The results will be more meaningful if the program is statically linked. This eliminates the spurious instructions used in linking a program to the libraries. Secondly, the pre-analyzer (sitapa) is run with spy and executable to extract a list of basic blocks and frequencies of the workload, which is then read by the control-dependence analyzer (sitadep) to produce an annotated list, as the third step. This annotations include control-dependency

relationships between the blocks and destination frequencies. Finally, the scheduler (sitarun) is run with the annotated list as input, and generally with spy and executable. The scheduler produces output indicating the parallelism available for the given input trace under the given oracle model. There are 69 basic instruction operations in SPARC. These instructions mainly fall into five basic categories: load/store, arithmetic/logic/shift, control transfer, read/write control register, and floating-point operate. Therefore, each parallel instruction presented by a vector of length five [23].

5.3 Experimental results

Due to the use of averages, it is important to show that for the represented workloads, the parallelism profile does not vary dramatically around its average parallelism. For that, we measure smoothability of the NPB workloads. Smoothability [12] is a metric designed to capture the parallelism profile variability around the average degree of parallelism. It is defined as the ratio of execution time with no restriction on the number of processors to the execution time when the number of available processors is limited to the average degree of parallelism. The interest in smoothability stems from the fact that the centroid is based upon the average degree of parallelism for each type of operation. Therefore, for centroids to well represent workloads, those workloads should have relatively high smoothability (close to 1). In this section we show that typical real-life applications, such as those represented by NPB, have high smoothability.

In Fig. 2 we list the parallelism results for the NAS Parallel Benchmark workloads running on the oracle model and present the smoothability values. Our results indicate that the parallelism obtained has a relatively smooth temporal profile which exhibits a high degree of uniformity in the parallelism except for the *cgm* benchmark whose smoothability is 68%. In all cases, but the *cgm* benchmark, the smoothability is better than 83%. Most importantly, in the context of this study, the smooth temporal behavior supports the fidelity of representing practical workloads using parallel instruction centroids.

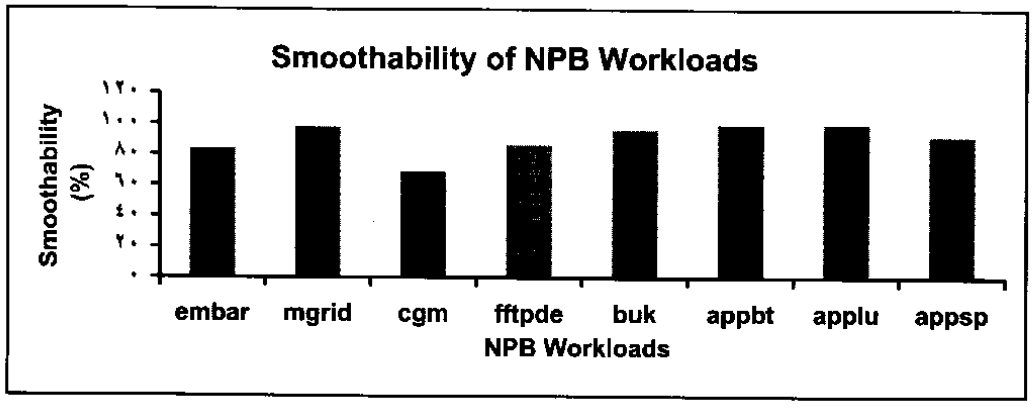

Fig. 2. The smoothability of the NAS parallel benchmark workloads.

Parallel-instruction centroid vectors can reveal differences in workload behavior that can not be distinguished by averages of parallelism degrees as shown in Table 7. Therefore, the parallelism behavior of two workloads can be efficiently compared by using the aforementioned parallel-instruction vector space model and the similarity function, expression (9), to quantify the similarity between these workloads.

Table 11. Centroid values for the NAS parallel benchmarks

Benchmarks	Intops	Memops	FPops	Controlops	Branchops
embar	81.344	59.469	14.369	0.000009	37.337
mgrid	33.857	19.516	0.7958	0.04973	9.22
cgm	4.475	3.798	0.84	0.000012	0.8463
fftpde	184.422	128.224	33.466	10.8513	57.765
buk	2.428	1.735	0.4502	0.000001	0.662
applu	1,031.789	559.136	69.79	0.04813	413.972
appsp	8,260.854	5,262.65	604.75	26.195	3,504.31
appbt	2,788.824	847.519	49.73	4.307	1,065.396

Table 12 quantifies the similarity between each pair of benchmarks in the NAS Parallel Benchmark suite, using expression (9). Again, note that the similarity in parallelism is not a transitive relation. We first compare *appsp* and *appbt*, two workloads that are representative of computations associated with the implicit operators of CFD codes such as ARC3D at NASA Ames. The relatively high value, 0.640, of the dissimilarity in parallelism illustrates that these two workloads have different parallelism behaviors. Next we consider *buk*, a workload representing the application area of integer sorting, and *cgm*, this workload is typical of unstructured grid computations.

Table 12. Similarity values for the nas parallel benchmarks

	embar	mgrid	cgm	fftpde	buk	applu	appsp	appbt
embar	0.000							
mgrid	0.530	0.000						
cgm	0.943	0.834	0.000					
fftpde	0.390	0.803	0.974	0.000				
buk	0.971	0.918	0.319	0.987	0.000			
applu	0.9066	0.967	0.9954	0.782	0.9976	0.000		
appsp	0.9895	0.9962	0.9994	0.9772	0.9997	0.8666	0.000	
appbt	0.966	0.987	0.998	0.924	0.999	0.4864	0.640	0.000

The relatively low value of the dissimilarity in parallelism behavior, 0.319, illustrates that these two workloads have relatively similar parallelism properties. Although the two workloads come from different application areas, each workload is expected to exercises target machines with a very similar mix of parallelism. The same conclusion of might be also drawn from the measurement, 0.390, of the dissimilarity in parallelism between *embar* and *fftpde* workloads.

Using the similarity of workloads, one can better interpret and predict performance measurements of a parallel benchmark suite on high-performance computing platform. This can be demonstrated by examining the NPB performance results from a recent report [20] in the light of Table 8. In this report, different parallel systems were used to run the NAS Parallel Benchmark suite. For example, let us compare the similarity between *embar* and *fftpde* benchmarks, 0.39, and the performance similarity results from the NAS report. The similarity in the performance geometric means [25] of these two applications is about 0.4896. For the case of the Cray-T3D only, it is about 0.4244. Another example is, *buk* and *cgm*. These two benchmarks have a similarity of 0.319, where the similarity in performance from the report is near to 0.3374. In the case of the Cray-T3D alone, the similarity is 0.2618. On the other hand, performance is less predictable for dissimilar benchmarks, as in *appbt* and *cgm*, see Tables 13 through 16.

Table 13. Workload and performance similarities for benchmarks embar and fftpde

SIM(embar, fftpde) = 0.39	
Machines	**Performance similarity**
Cray-T3D	0.4244
IBM SP-2	0.4098
Kendall Square KSR1	0.3251
Meiko CS-2	0.3651
MasPar MP-2	0.3606
SG/Power Challenge XL	0.4218

Table 14. Workload and performance similarities for benchmarks buk and cgm

SIM(buk, cgm) = 0.319	
Machines	**Performance similarity**
Cray-T3D	0.2618
Fujitsu VPP500	0.3988
IBM SP-1	0.2849
IBM SP-2	0.2827
Kendall Square KSR2	0.2947
MasPar MP-2	0.2752
nCUBE-2S	0.3874
Intel Paragon (OSF1.2)	0.3874

Table 15. Workload and performance similarities for benchmarks mgrid and embar

SIM(mgrid, embar) = 0.53	
Machines	**Performance similarity**
Cray-T3D	0.5181
Intel iPSC/860	0.4746
Kendall Square KSR1	0.5694
Kendall Square KSR2	0.5836
Meiko CS-1	0.5371
Thinking Machines CM5E	0.5493

Table 16. Workload and performance similarities for benchmarks appbt and cgm

SIM(appbt, cgm) = 0.998	
Machines	**Performance similarity**
Cray-T3D	0.8558
Thinking Machines CM-5	0.7538
MasPar MP-1	0.4546
nCUBE-2S	0.4881

6. Conclusions

This paper introduced a methodology for parallelism-based representation of workloads. The method is architecture-invariant and can be used effectively for the comparison of workloads. A comparative study between the parallelism-matrix technique and our parallel-instruction vector space model was also presented. It was shown that the parallelism-matrix technique depends only on identical rather than similar parallel instructions. However, the introduced parallel-instruction vector space model takes all parallel instructions into account when representing workloads and their similarities. Furthermore, while the parallelism-matrix technique requires $O(p\,t)$ computational time for workload representation, the parallel-instruction vector space model requires only $O(t)$. Considering the storage requirements, the parallelism-matrix technique needs $O(n^t)$ memory space, whereas the parallel-instruction vector space model needs only $O(t)$. In addition, when two workloads are compared, the computational cost in the parallelism-matrix technique is $O(n^t)$. On the other hand, the parallel-instruction vector space only requires $O(t)$ computational time. Hence, the parallel-instruction vector space model does not only provide more accurate, but also more cost-effective parallelism-based representation of workloads.

The parallel-instruction workload model was used to study the similarities among the NAS Parallel Benchmark workloads in a quantitative manner. The results confirm that workloads in NPB represent a wide range of parallel instruction mixes. Further, the model was also used to shed some light on the predictability of how one application from NPB performs, given that the performance of a similar workload is known.

References

[1] Kuck, D. J., Muraoka, Y. and Chen, S. C. "Measurements of Parallelism in Ordinary FORTRAN Programs." *Computer*, Vol. 7 (1974), 37-46.

[2] Riseman, E. M. and Foster, C. C. "The Inhabitation of Potential Parallelism by Conditional Jumps." *IEEE Transactions on Computers*, C-21, 12 (December 1972), 1405-1411.

[3] Tjaden, G. S. and Flynn, M. J. "Detection and Parallel Execution of Independent Instructions." *IEEE Transactions on Computers*, C-19, No. 10 (October 1970), 889-895.

[4] Nicolau, A. and Fisher, J.A. "Measuring the Parallelism Available for Very Long Instruction Word Architectures." *IEEE Transactions on Computers*, 33, No. 11 (Nov. 1984), 968-976.

[5] Butler, M., Yeh, T., Patt, Y., Alsup, M., Scales, H. and Shebanow, M. "Single Instruction Stream Parallelism is Greater Than Two." *In: Proceedings of the 8th Annual Symposium on Computer Architecture*, (May 1991), 276-286.

[6] Lam, M. S. and Wilson, R. P. "Limits of Control Flow on Parallelism." *In: Proceedings of the 19th Annual International Symposium on Computer Architecture*, Australia: Gold Coast, (May 19-21, 1992), 46-57.

[7] Kumar, M. "Measuring Parallelism in Computation-Intensive Scientific/Engineering Applications." *IEEE Transactions on Computers*, C-37, No. 9 (Sep. 1988), 1088-1098.

[8] Wall, D.W. "Limits of Instruction-level Parallelism."*Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, California: Santa Clara, (April 8-11, 1991), 176-188.

[9] Austin, T. and Sohi, G. "Dynamic Dependence Analysis of Ordinary Programs," *Proceedings of the 19th Annual International Symposium on Computer Architectures*, (1992) 342-351.

[10] Arvind, Culler, D. E. and Maa, G. K. "Assessing the Benefits of Fine-Grained Parallelism in Dataflow Programs." *Proceedings of Supercomputing* 88, (November 1988), 60-69.

[11] Uht, A. K. "Extraction of Massive Instruction Level Parallelism." *ACM SIGARCH News*, (June 1993), 5-12.

[12] Theobald, K. B., Gao, G. R. and Hendren, L. J. "On the Limits of Program Parallelism and Its Smoothability." *Proceedings of the 25th Annual International Symposium on Micro-architecture* (MICRO-25), Portland: Oregon, (December 1992) 10-19.

[13] Conte, T. and Hwu, W. "Benchmark Characterization." *IEEE Computer*, (Jan. 1991),48-56.

[14] Calzarossa, M. and Serazzi, G. "Workload Characterization for Supercomputer." Performance Evaluation of Supercomputers, J. L. Martin (Ed.), North-Holland, (1988), 283-315.

[15] Martin, J. "Performance Evaluation of Supercomputers and Their Applications." Parallel Systems and Computation, Paul, G. and Almasi, G. (Eds.), North-Holland, (1988), 221-235.

[16] Lee, J.K.F. and Smith, A.J. "Branch Prediction Strategies and Branch Target Buffer Design." *Computer*, 17, No. 1 (January 1984), 6-22.

[17] Hockney, R. *The Science of Benchmarking*. Tutorial Handouts, Supercomputer 94, Washington DC, (November 1994).

[18] Bradley, D. and Larson, J. "A Parallelism-based Analytic Approach to Performance Evaluation Using Application Programs." *Proceedings of the IEEE*, 81, No. 8 (August 1993). 1126-1135.

[19] Meajil, Abdullah I., El-Ghazawi, Tarek and Sterling, Thomas "A Quantitative Approach for Architecture-Invariant Parallel Workload Characterization." *In: Lecture Notes in Computer Science*, No. 1184, Wasniewski, J. Dongarra, J. Madsen, K. and Olesen, D. (Eds.), Applied Parallel Computing: Industrial Computing and Optimization, Proceedings of the *Third International Workshop, PARA '96*, Denmark: Lyngby, (August 18-21,1996), 515-524.

[20] Bailey, D., Barszcz, E., Dagum, L. and Simon, H. "NAS Parallel Benchmark Results 10-94." *NAS Technical Report NAS-94-001*, (October 1994).

[21] Bailey, D. *The Science of Benchmarking*. Tutorial Handouts, Supercomputer 94, Washington DC, (November 1994).

[22] Bailey, D. *et al*, "The NAS Parallel Benchmarks." *Intl J. Supercomputer Applications*, 5, No. 3 (1991), 63-73.

[23] Bailey, D., Barszcz, E., Dagum, L. and Simon, H. "NAS Parallel Benchmark Results." *IEEE Parallel & Distributed Technology*, (February 1993), 43-51.

[24] Fleming, P. and Wallace, J. "How Not to Lie with Statistics: The Correct Way to Summarize Benchmark Results." *Communications of ACM*, 29, No. 3 (March 1986), 218-221.

[25] Meajil, Abdullah I. and El-Ghazawi, Tarek "Workloads and Performance Similarities in Parallel Architecture Benchmarks." *Proceedings of the Conference on Communication Networks and Distributed Systems Modeling and Simulation, CNDS '97*, Lionel Ni and Taieb Znati, (Eds.), Phoenix, AZ, (January 12-15, 1997), 31-36.

نموذج فضاء شعاعي لتشخيص العبئ خلال العمل المتوازي

عبدالله المعجل، طارق الغزوي* و ثوماس سترلنج**

*قسم الهندسة الكهرباء و علوم الحاسب، جامعة جورج واشنطن.

** مركز قاعدة فضاء شعاعي و نظم معلومات (CESDIS)

مركز ناسا قودارد، الولايات المتحدة الأمريكية

(قدم للنشر في ١٩٩٥/١٢/٢٧م؛ وقبل للنشر في ١٩٩٦/١١/٢٦م)

ملخص البحث. تعتبر البرمجة المتوازية عاملا رئيسيا في أداء الأنظمة المتوازية. نناقش في هذه الورقة نموذج فراغي شعاعي ذي أوامر برمجية متوازية لتمثيل الحمل ومقارنته. وسوف يتم مقارنة هذا النموذج مع أسلوب المصفوفة المتوازية المعتمدة على قانون فروبنيوس المصفوفي. ويقارن الأسلوب الأخير المذكور حملين معتمدين على جميع الأوامر البرمجية المتوازية. وسوف يتم إثبات أن الأسلوب الفراغي الشعاعي والذي يتميز بالأوامر البرمجية المتوازية يفوق أداؤه أسلوب المصفوفة المتوازية في الزمن والفراغ ، كما يفوقه في الدقة. وسوف يتم أيضا إثبات أن هذا النموذج المذكور يؤمن إطار مفيد للتصميم وتحليل الخصائص . وسوف يتم إظهار هذا بواسطة تحليل بعض الخصائص من نوع NASA/NAS للأحمال وتحليل قياسات أدائها في إطار هذا النموذج المدروس .