

## **Characterization of ILP Distribution for NASA NAS Parallel Benchmarks**

**Abdullah I. Almojel**

*Department of Computer Engineering  
King Fahd University of Petroleum and Minerals,  
P.O. Box 2045, Dhahran 31261, Saudi Arabia*

(Received 26 January 2003; accepted for publication 29 July 2003)

**Abstract.** A characterization study of analyzing dynamic instruction traces to characterize program parallelism is conducted. This study supports that the experimental design of supercomputer and parallel computers calls for quantifiable methods to evaluate the requirements of different workloads within an application domain. Such methods can help establish the basis for scientific design of parallel computers driven by application needs, to optimize performance to cost. In addition, the application characteristics can be used early in the design process to identify bottlenecks such as not having enough resources, not having enough parallelism in the instruction stream, or using a too restrictive scope of concurrency detection. The selection of which features to include in a new computer system depends on the needs of the workloads the system will execute. The number and type of functional units are among the most important design decisions. Therefore, this paper presents an instruction-level characterization for analyzing dynamic traces using a trace-driven simulator. It investigates the parallel system needs for a class of contemporary benchmarks taken from NASA/NAS Parallel Benchmarks (NPB) suite. The NPB represents an implementation independent problem set, representative of Computational Aerospace workload computations. The NPB workloads have been implemented on nearly every parallel platform and results have been reported by the vendors. Data is presented for NPB requirements of these resources. The requirements suggest upper limits on the resources needed for efficient processors. In this study, we also examine non-uniformities in the distribution of instruction-level parallelism. Several non-uniformities in instruction-level parallelism are investigated including variation between benchmark class and by instruction class within benchmark. In addition, the average instruction class distribution as well as the shortest path a workload would be executed on a parallel machine will be shown. The results confirm that workloads in NPB represent a wide range of non-redundant applications with different characteristics.

**Keywords:** Instruction-Level Parallelism; Workload Characterization; Benchmarking NASA NPB; Parallel Processing; Smoothability.

### **1. Introduction**

The advances of computer architectures are generally affected by two things: a better understanding of program execution, and new or better implementation technologies. It

is therefore very important to understand the dynamics of program execution when considering the design of future-generation architectures. Hence, experimental design of parallel computers calls for quantifiable methods to evaluate the requirements of different workloads within an application domain. Such methods can help establish the basis for scientific design of parallel computers driven by application needs, to optimize performance to cost [1, 13, 15, 16]. The selection of which features to include in a new computer system depends on the needs of the workloads the system will execute. Also, a critical factor in the design and development of high-performance computing architectures and applications is to understand and to characterize their performance and the corresponding requirements of computing and network resources [11, 15, 16].

It follows quite natural (and has been observed in the history of performance evaluation) that changes in the computing environment (parallel and distributed computing, network computing, mobile computing, etc.) and new system features (security and reliability mechanisms, agent-based systems, intelligent networks, adaptive systems, etc.) pose new challenges to performance evaluation by raising the need for new analysis methodologies. From a load modeling point of view, the difference in using computing resource has changed the type of model for workload characterization. While in the early days of computing (70s) the typical systems were used in batch or interactive mode, static workload models could adequately represent the user behavior. In the 80s, dynamic workload models were introduced, which were able to represent variabilities in user behavior. Within the last years, generative workload models have been proposed as a suitable method for bridging the gap between the user's application oriented view of the load and the actual load (physical, resource oriented requests) submitted to the system [9, 10].

Therefore, the viability of a parallel processing system depends heavily on the presence of parallelism in the application programs expected in the workload of that system. Since the scientific applications domain designated as the primary beneficiary of parallel processing, there have been several attempts to make parallel machines targeted for scientific applications. In each case, the claim for the optimality of the machine design is based on the assumptions about the amount of parallelism present in the applications and its nature. In this paper, the Numerical Aerodynamic Simulation (NAS) Parallel Benchmarks (NPB) developed at NASA Ames [2, 14] are used in order to quantify their resource needs for parallel processing. The NPB represents an implementation independent problem set, representative of Computational Aeroscience workload computations. The requirements suggest upper limits on the resources needed for efficient processors. In this study, we also examine non-uniformities in the distribution of instruction-level parallelism. Several non-uniformities in instruction-level parallelism are investigated including variation between benchmark class and by instruction class within benchmark. We also consider the behavior of programs under the constraint that at most a certain number of operations can be performed simultaneously

[15]. We use a metric called "smoothability" [3] to show how evenly parallelism can be distributed. Therefore, smoothability is a metric designed to capture the parallelism profile around the average degree of parallelism. This metric also quantifies how efficiently a parallel program can run on a multiprocessor with limited processors.

The parallel execution model of our study is based on a dataflow model where any two operators can execute in parallel, unless one actually provides data directly or indirectly to the other. We decided to base the analysis on traces of workloads executed on Sun SparcStations [4] at NASA Goddard Space Flight Center. The Sparc architecture was chosen for the testbed because it is representative of present-day RISC processors.

This paper is organized as follows: next section presents the parallel execution model of the study. The experimental tools used in this study will be presented in section 2, too. Section 3 provides an overview of the NAS Parallel Benchmarks suite. Section 4 shows the methodology and the analysis process. Sections 5 and 6 present and analyze some of the experimental results conducted in this study. Finally, conclusions are in section 7.

## 2. NAS Parallel Benchmarks

The NAS Parallel Benchmarks (NPB) suite, which is an interesting alternative to traditional benchmarking suites, were devised by the Numerical Aerodynamic Simulation (NAS) Program of the National Air and Space Administration (NASA) for the performance evaluation and analysis of highly parallel supercomputers. While the NPB suite is rooted in the problems of computational fluid dynamics and computational aerodynamics, they are valuable in the evolution of parallel computing [12], since they are rigorous and as close to real applications as may be reasonably expected from a benchmarking suite. Therefore, using the NPB suite is a great deal more difficult than using something like SPEC, because it involves writing a set of tuned parallel applications. However, the suite gives manufacturers a chance to demonstrate what their machines can do in a way that is impossible with more traditional benchmarks [9, 12].

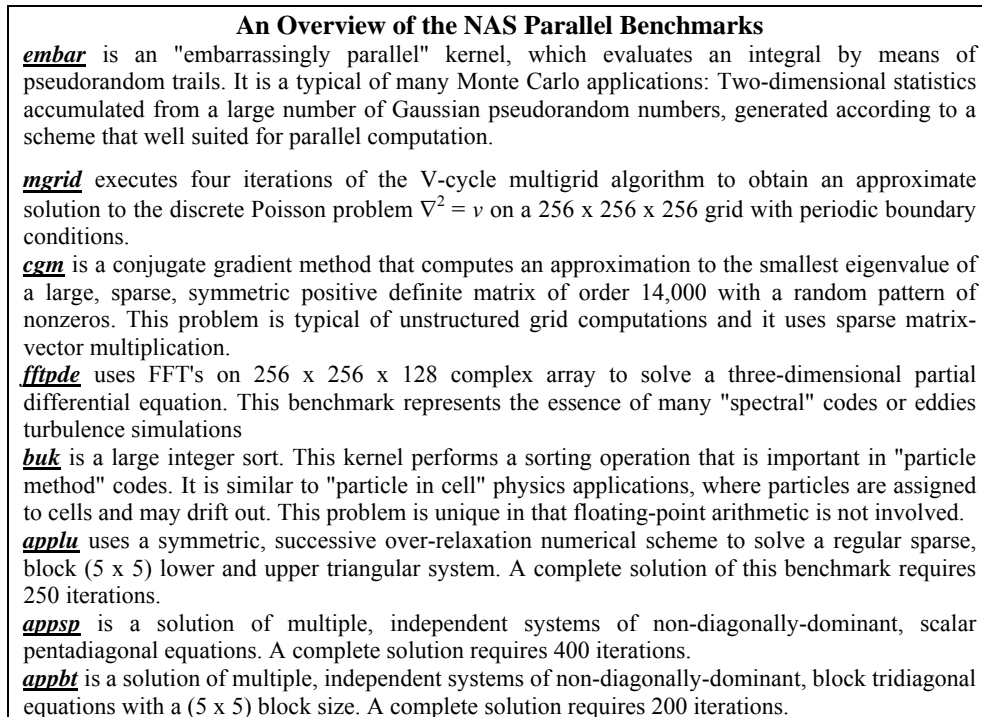
The NPB is a set of eight benchmark problems that are designed to measure the sustained performance of highly parallel computer systems for a subset of algorithms that characterize various computationally intensive aerophysics applications. The eight problems consist of five kernels and three simulated computational fluid dynamics (CFD) application benchmarks. This benchmark suite successfully addresses many of the problems associated with benchmarking parallel machines. It represents the principal computational and data movement requirements of modern CFD applications. Table 1 lists the numerical problems solved in each benchmark (the embar and buk benchmarks are not listed in this table because they are non-numerical benchmarks). The following

Table 1 describes each benchmark along with the architectural properties exercised by the benchmark. It should be noted that NPB is a “paper and pencil” benchmark that specifies the benchmarks only algorithmically. An exhaustive description of these NPB problems is given in [2, 14].

**Table 1. Numerical problems solved in the NAS parallel benchmarks**

| Numerical problem            | Benchmark   |
|------------------------------|---|
| sparse linear system solvers | <i>cgm</i> : conjugate gradient benchmark<br><i>lu, sp, bt</i> : simulated CFD applications |
| fast Fourier transforms      | <i>fftpde</i> : 3-dimensional FFT benchmark   |
| elliptic problem solvers     | <i>lu, sp, bt</i> : simulated CFD applications  |
| multigrid schemes            | <i>mgrid</i> : 3-dimensional multigrid benchmark  |

The following Fig. 1 gives an overview of the NPB problems. The first five are the parallel kernel benchmarks, and the last three are the simulated CFD application benchmarks intended to accurately represent the principal computational and data movement requirements of modern CFD applications.



**Fig. 1. An overview of the NAS parallel benchmarks.**



### 3. Parallel Execution Model and Measurement Tools

To be universally applicable, the system characterization measurements must be based on a uniform model of execution for parallel computations so that the results of an experiment can be related to previous and future experiments. Hence, the architecture undergoing this study would be the *oracle model* architecture.

#### 3.1 Parallel computation model

This model presents the most ideal machine that has unlimited processors and memory (infinite resources). It does not incur any overhead in scheduling tasks and managing machine resources, does not incur any communications (with 0-cycle cost) and synchronization overheads, and detects and exploits all the parallelism present in a program (no resources/synchronization induced constraints on parallelism). This machine should have perfect memory disambiguation (a language which resolves ambiguities), unlimited register and memory renaming (with single-cycle latency), no barriers due to control dependencies, perfect branch prediction, and no window size limitations. Thus, this abstract machine model is called the oracle model [3, 5].

The Sparc architecture instruction set [4] was chosen for the testbed because it is representative of present-day RISC processors. Sparc is a CPU instruction set architecture (ISA), derived from a reduced instruction set computer (RISC) lineage. As an architecture, Sparc allows for a spectrum of chip and system implementations at a variety of price/performance points for a range of applications, including scientific/engineering, programming, real-time, and commercial. A Sparc processor logically comprises an integer unit, a floating-point unit, and an optional coprocessor, each with its own registers. This organization allows for implementations with maximum concurrence between integer, floating-point, and coprocessor instruction execution.

Instructions in Sparc are accessed by the processor from memory and are executed, annulled, or trapped. Instructions are. There are 69 basic instruction operations encoded in three 32-bit formats and can be partitioned into five general categories: load/store, arithmetic/logical/shift, control transfer, read/write register, and floating-point operate instructions. More details about the Sparc instruction set are found in [4].

#### 3.2 Measurement and experimental tools: Sparc performance analyzer package tool

The Sparc Performance Analysis (Spa) package is a set of tools which run on Sparc systems and are used to analyze the performance of Sparc binary application programs on the Sparc station-1 and Sparc station-2 [6]. The Spa package can be used on any Sun4 architecture machine running a SunOS 4 (preferably 4.1) operating systems. The main component of the Spa package is a Sparc simulator and trace generator. The

Spa tools include *spy* a program that traces the execution of a command and can generate spat address traces, *spanner* a tool that converts an spat address trace into spic instruction count files, *splice* a tool that combines spic instruction count files, and *spout* a tool that displays a spic instruction count.

However, the Spa package does not predict overall system performance. I/O latency is not taken into consideration, nor is the effect of more than process being active at any one time. Although Spa has some other deficiencies as well, it includes tool that is shown to be convenient to build around: *spy* that traces the execution of a command. It can be used to generate an address trace which can then be passed directly to a trace analyzer-it runs about 600 times slower than normal, and/or list the system calls performed by that command. For each system call, the attributes of the process being traced, the address of the system call, the name of the routine being called, and some other useful information are extracted.

#### **Sequential instruction trace analyzer tool**

The Sequential Instruction Trace Analyzer (Sita) is a tool package developed at the McGill university to measure the amount of parallelism which theoretically exists in a given workload [3]. Sita takes a dynamic trace generated 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. This architecture can vary according to parameters set by the user, who can select such features as the number of processors or the method of branch prediction used. This allows comparisons between competing architectures that use specific benchmarks to be placed in context, shows designers a glimpse of how much parallelism can be exposed by adding various features to their machines, and gives algorithm designers a tool for comparing implementations of alternative algorithms.

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 [6]. Sita tool includes a pre-analyzer (*sitapa*), a control-dependence analyzer (*sitadep*), and a trace scheduler (*sitarun*). Sita tools are mentioned below in the order they are used to extract parallelism figs:

*Sita pre-analyzer (sitapa)*: is an auxiliary tool providing a total of four functions that can be desired. Its primary function is to analyze a dynamic trace to determine the boundaries of the basic blocks (a basic block is a sequence of instructions that is always entered at the beginning and exited at the end without any branch or jump), and to record the frequencies of branch outcomes. This function will also record some useful statistics about the trace: the total number of executed and annulled instructions; the number of memory accesses; the number of floating-point operations; deepest depth reached at the

stack; the total number of basic blocks in the workload, both static in the workload and dynamic in the trace.

*Sita control-dependent analyzer (sitadep)*: performs control-dependence analysis on a list of basic blocks which is generated by sitapa. It produces an annotated list of basic blocks and control-dependency information which consist of , among others, the total number of procedures, the number of instructions in the longest basic block, the most common distinctions from the blocks, and the critical blocks (those on which other blocks are control dependent).

*Sita trace scheduler (sitarun)*: reads in a dynamic trace and schedules the instructions according to an idealized parallel machine(s) based on user-definable parameters. The total instruction level of parallelism is measured and reported along with the description of the model and some statistics on prediction rates and speculation depth. The parallelism is computed by dividing the total number of useful sequential instructions (the work) by the number of parallel instructions needed to schedule these sequential instructions (the time), which are both given in the output. Certain options may cause sitarun to output additional information.

#### 4. Methodology and Analysis Process

Characterization of a workload requires the measurement of its runtime behavior. The behavior we measure is in terms of an idealized architecture. This abstract system is general enough to include many system designs as special cases and then measure workload performances in terms of this abstract system, whose instruction set is the compiler intermediate code. These instructions are interpreted by an oracle model simulation, and the results are used to produce the workload characteristics. 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 while respecting all relevant dependencies between instructions.

In our NAS Parallel Benchmarks characterization study, trace analysis begins with a trace of the execution of a benchmark on the oracle model architecture. This trace consists of a stream of operations representing the actual order of instructions executed (not the static object code). For each executed operation, the trace gives the opcode, PC address, memory-access address (if any), and the destination of a branch or jump. Thereafter, the analyzer reads the operations from the stream and schedules them according to the oracle model. As the analyzer reads each operation in the trace, it inserts the operation into the earliest parallel instruction possible, while simultaneously respecting the dependencies between that operation and all operations. Thus, the scheduled instructions are packed into parallel instructions.

Our analysis process of a workload takes four steps. First, a Sparc executable file is created, using the desired optimization level. All workloads are compiled with optimization using the Fortran compiler, *f77*, of SunOS 4.1.1. 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. Then a dynamic execution trace of the workload is captured using the *spy* tool. This trace fed directly to the pre-analyzer (*sitapa*) 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. If *spy* is used with a trace analyzer, such as *sitapa* or *sitarun*, the system will run 400-600 times slower than normal.

## 5. Measurements and Experimental Results

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 Fortran-77 codes implementing the benchmarks, provided by NAS, actually solve scaled-down versions of the NPB problems that run on many current-generation high performance workstations. The standard input sizes for the NPB suite referred to as the Class A and Class B size problems. Table 2 lists the problem size [2] 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 [6] on a Sun workstation at NASA Goddard Space Flight Center.

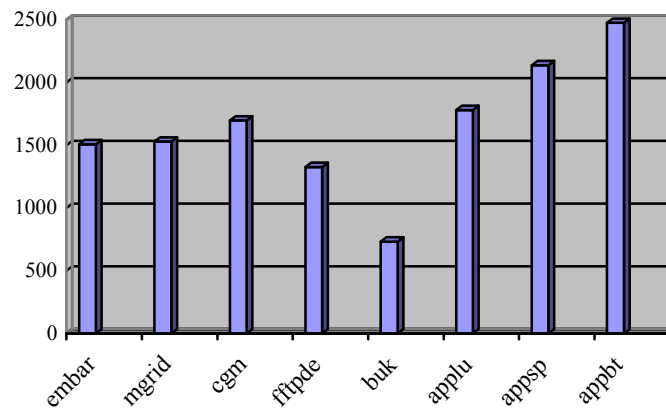
Note that in the case of *mgrid* the grid size is unchanged, but a greater dynamic operation count results from changes in the inner loop iterations. An explanation of the entries in the problem size column found in the corresponding sections describing the benchmarks in [2].

**Table 2. Dynamic operation counts for NPB running on a sparc processor**

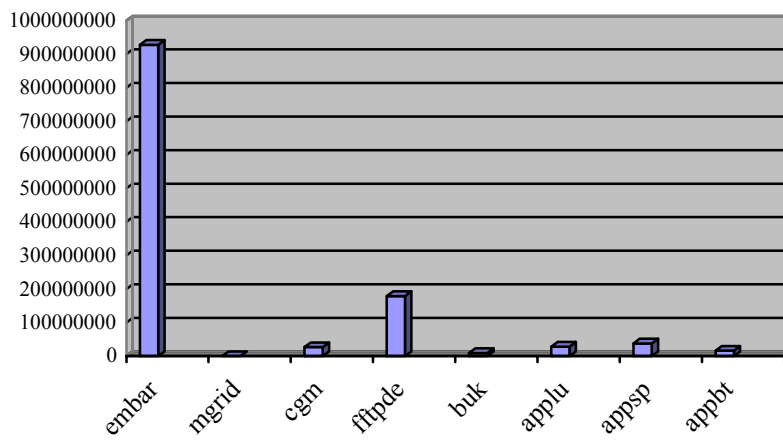
| Benchmark     | Problem size   |                        |                        | Dynamic operation count ( $10^9$ ) |         |         |
|---------------|----------------|------------------------|------------------------|------------------------------------|---------|---------|
|               | Sample         | Class A                | Class B                | Sample                             | Class A | Class B |
| <i>Embar</i>  | $2^{24}$       | $2^{28}$               | $2^{30}$               | 8.9811                             | 26.68   | 1008.8  |
| <i>Mgrid</i>  | $32^3$         | $256^3$                | $256^3 \times 7$       | 0.1154                             | 3.905   | 18.81   |
| <i>Cgm</i>    | $\approx 10^5$ | 14,000                 | 75,000                 | 0.5103                             | 1.508   | 54.89   |
| <i>Fftpde</i> | $64^3$         | $256^2 \times 128$     | $256^2 \times 512$     | 1.5230                             | 5.631   | 71.37   |
| <i>Buk</i>    | $2^{16}$       | $2^{23} \times 2^{19}$ | $2^{25} \times 2^{21}$ | 0.0768                             | 0.7812  | 3.150   |
| <i>Applu</i>  | $12^3$         | $64^3$                 | $102^3$                | 0.5200                             | 64.57   | 319.6   |
| <i>Appsp</i>  | $12^3$         | $64^3$                 | $102^3$                | 0.8920                             | 102.0   | 447.1   |
| <i>Appbt</i>  | $12^3$         | $64^3$                 | $102^3$                | 1.1157                             | 181.3   | 721.5   |

† Code is different from Class A [2].

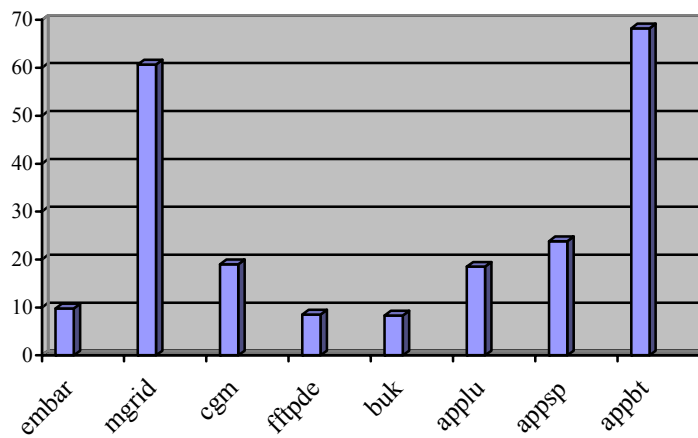
There exists a number of parameters for characterizing programs, including instruction class mix, average basic block length, and dynamic instruction trace length. Static and dynamic basic block counts and average basic block lengths of NPB workloads are shown in Graphs 1, 2, and 3, respectively. Quite a disparity exists between the average basic block lengths of the benchmarks from a high of about 68 for *appbt* to a low of 8 for *buk*. Tables 3 and 4 show the total number of dynamic sequential and parallel instructions exhibited by the different NPB workloads for Sample and Class A input sizes, respectively. In addition, Graphs 4 and 5 exhibit the average degree of parallelism of the NPB workloads for Sample and Class A input sizes, respectively. The instruction class mix for NPB is presented in Graph 6 for Sample input size and Graph 7 for Class A input size. In the oracle machine, parallel instruction count is also the critical path length. Useful instruction count sums all the dynamic sequential instructions that are not annulled or delay operations. However, the dynamic instruction counts as well as the degree of parallelism for some of the NPB workloads with Class A input size have been extrapolated due to the limitation of memory space. Also, the average degrees of parallelism of NPB workloads are presented in Graphs 4 and 5 for Sample and Class A input size, respectively.



Graph 1. Static basic block count for sample of NAS parallel benchmarks.



Graph 2. Dynamic basic block count for sample of NAS parallel benchmarks.

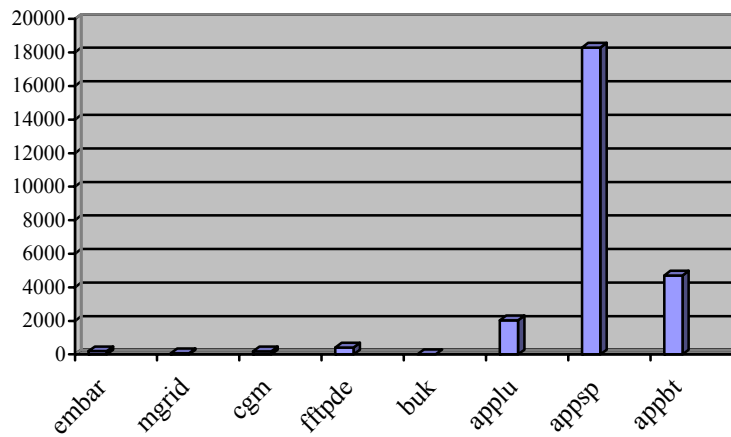


Graph 3. Average basic block length for sample of NAS parallel benchmarks.



**Table 3. Sequential and parallel instruction counts for the NPB sample**

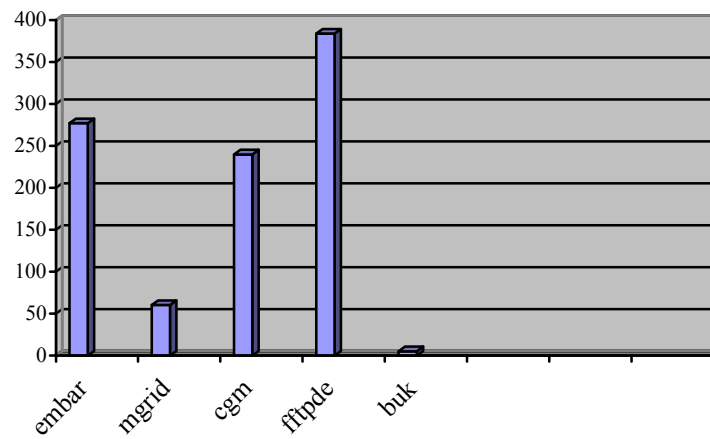
| Benchmark     | Useful instructions | Parallel instructions |
|---------------|---------------------|-----------------------|
| <i>Embar</i>  | 9,042,071,827       | 46,650,326            |
| <i>Mgrid</i>  | 114,729,462         | 1,818,865             |
| <i>Cgm</i>    | 508,804,545         | 2,697,227             |
| <i>Fftpde</i> | 1,503,948,608       | 3,672,249             |
| <i>Buk</i>    | 72,576,990          | 14,553,108            |
| <i>Applu</i>  | 516,405,728         | 253,377               |
| <i>Appsp</i>  | 874,045,757         | 47,804                |
| <i>Appbt</i>  | 1,114,870,145       | 237,225               |



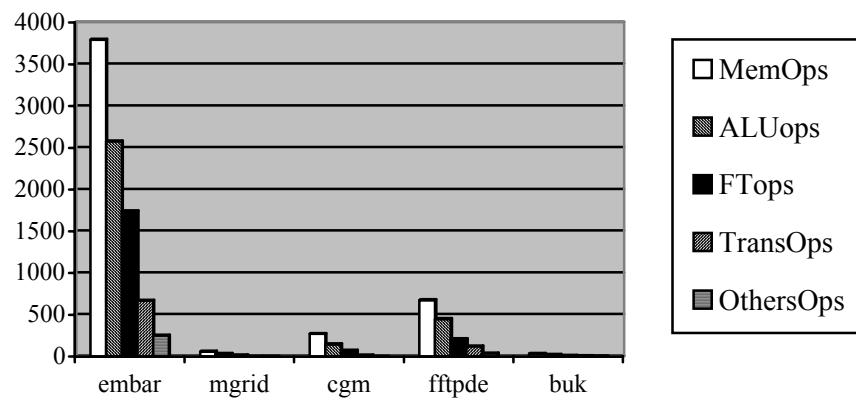
**Graph 4. Average degree of parallelism for the NPB sample.**

**Table 4. Sequential and parallel instruction counts for the NPB class A**

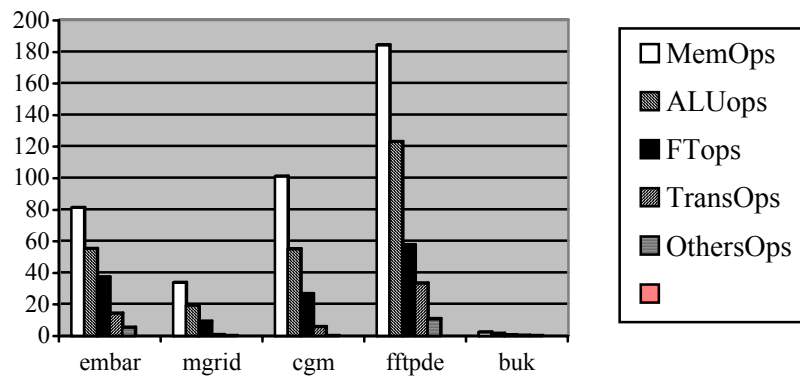
| Benchmark     | Useful instructions | Parallel instructions |
|---------------|---------------------|-----------------------|
| <i>Embar</i>  | 144,673,149,700     | 522,192,548           |
| <i>Mgrid</i>  | 56,042,127,780      | 928,827,554           |
| <i>Cgm</i>    | 825,571,323         | 3,440,844             |
| <i>Fftpde</i> | 22,559,399,470      | 58,755,984            |
| <i>Buk</i>    | 9,738,763,264       | 1,862,852,960         |



Graph 5. Average degree of parallelism for the NPB class A.



Graph 6. Operation counts for each instruction type in the NPB "sample" in millions.



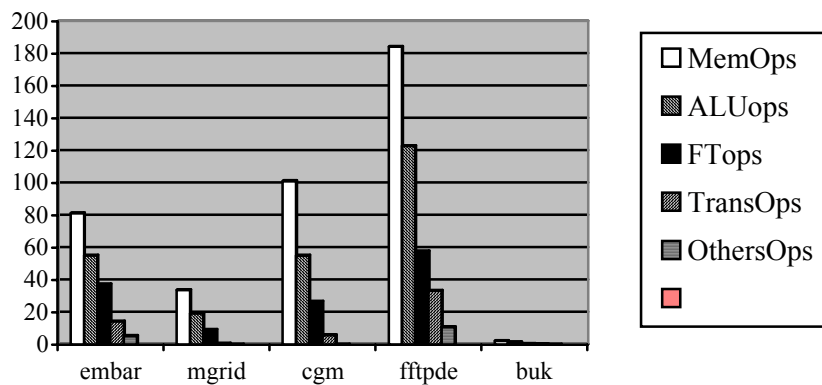
Graph 7. Operation counts for each instruction type in the NPB “class A” in millions.

## 6. Non-uniformities in Instruction-level Parallelism

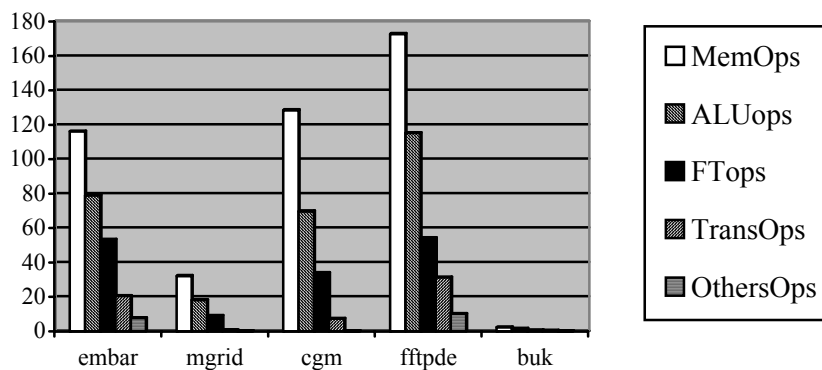
This section examines non-uniformities in the distribution of instruction-level parallelism. The non-uniformities in instruction-level parallelism include variations between workloads and variations by instruction class. The first non-uniformity in instruction-level parallelism we considered is variation in instruction-level parallelism from workload to workload. The NPB workloads actually have different amounts of instruction-level parallelism. The performance improvement in each workload when executed on an ideal architecture, *oracle model*, of unlimited parallelism is given in Graphs 4 and 5 for Sample and Class A input size, respectively. *buk* benchmark has the least amount of instruction-level parallelism whereas *apbt* has the most amount of instruction-level parallelism. Although similar benchmarks will exhibit similar average parallelism on a given machine, the converse is not true; similar average parallelism does not indicate that benchmarks are similar [7, 13]. This is because such benchmarks could be requiring different types of resources for their parallel operations. Graphs 4 and 5 show the average degree of parallelism for the NAS benchmarks with Sample and Class A input size, respectively.

Graphs 8 and 9 show the average parallelism within typical instruction classes. This data was obtained by simulating an unlimited issue oracle machine. Characterizing the inherent parallelism in programs is important insofar as it allows us to infer the behavior of the program on a realistic machine, i.e., a machine with finite number of processors. However, average parallelism can be a misleading indicator of performance. In this section, we focus on the question of whether a given program has sufficient

parallelism for a  $p$  processor machine, where  $p$  equals the average degree of parallelism in a benchmark. We present a technique to characterize the behavior of programs on a finite number of processors by modifying the oracle machine to constrain the program to execute a finite number of operations in each step. Therefore, smoothability [3] 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 first step towards a realistic execution model, a limited of  $p$  operations is imposed to be executed during each step.

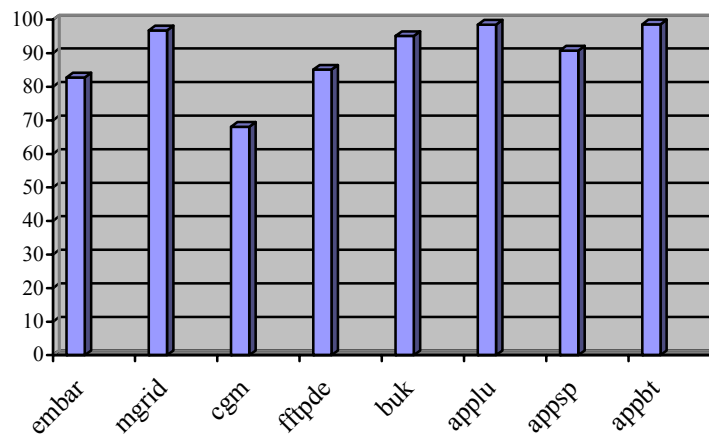


Graph 8. Average parallelism within instruction classes for the NPB "sample".



**Graph 9. Average parallelism within instruction classes for the NPB “class A”.**

In Graph 10 we list the parallelism results for the NAS Parallel Benchmark workloads running on the oracle model architecture with one restriction: the maximum number of processors are limited to the higher integer of the average degree of parallelism, 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%.

**Graph 10. The smoothability values (%) for NAS parallel benchmarks.**

## 7. Conclusion

This paper characterizes the structure and resource requirements of the NAS Parallel Benchmarks (NPB), a popular benchmark suite used to evaluate various parallel computers. The model is used to obtain parameter values. These quantitative parameters are useful in the design and evaluation of various parallel computers. In this paper, we have presented a method for quantifying the parallelism in real programs developed in the context of a dataflow model. It allows programs to be studied in full detail, without biasing their behavior by implementation constraints.

This allows us to draw a clear distinction between the parallelism inherent in a program and the speedup achieved under any specific implementation. In this study we have investigated the instruction-level parallelism for the NAS Parallel Benchmarks. The

oracle model architecture was used as the parallel execution model with the Sparc instruction set for our study. Different input sizes were used to produce the experimental results: the Sample and Class A input size problems. This paper has also presented design information useful for designs high-performance processors. A method for selecting function unit needs based on unlimited resource usage simulation was presented and suggested design parameters.

We have measured the length of the critical path of computation through the programs, and measured the average parallelism. These studies indicated that there is a useful amount of parallelism in most of benchmarks. Several non-uniformities in instruction-level parallelism were investigated including variations between benchmarks and by instruction class within benchmark. The results confirm that workloads in NPB represent a wide range of non-redundant applications with different characteristics. This study has shown that typical real-life applications, such as those represented by NPB, have high smoothability.

**Acknowledgment:** I would like to acknowledge the King Fahd University of Petroleum and Minerals for their support. This work has been supported by the Center of Excellence in Space Data and Information Sciences at NASA Goddard Space Flight Center under Grant No. NAS5-30428. This work has been supported by NASA High-performance Computing and Communications (HPCC) program through CESDIS/USRA, Grant No. NAS5-30428.

### References

- [1] Meajil, A.I., El-Ghazawi, T. and Sterling, T. "A Quantitative Approach for Architecture-Invariant Parallel Workload Characterization." Workshop on Applied Parallel Computing in Industrial Problems and Optimization (PARA'96), Lyngby, Denmark (August, 1996), 18-21.
- [2] Bailey, D. et al. *The NAS Parallel Benchmarks*, RNR Technical Report RNR-94 007, March 1994, NASA Ames Research Center, Moffett Field, CA.
- [3] 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.
- [4] The SPARC Architecture Manual. Version 8, Menlo Park, CA: SPARC Int'l, Inc., 1991.
- [5] 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.
- [6] Irlam, G. *The Spa Package*, Version 1.0, October 1991.
- [7] Meajil, A.I. "An Architecture-Independent Workload Characterization Model for Parallel Computer Architectures." *Technical Report No. GWU-IIST 96-12, Department of Electrical Engineering and Computer Science*, George Washington University, July 1996.
- [8] Almojel, A. "Numerical Aerodynamic Simulation Parallel Benchmarks: A Characterization Study." *The Eighth SIAM Conference on Parallel Processing for Scientific Computing*, Minneapolis, MN, SIAM Activity Group on Supercomputing, Order Code # PR 0094. (March 14-17, 1997), 97

- [9] Haring Günter, Kosits Gabriela and Raghavan, S.V. "Workload Characterization in High-performance Computing Environments." *Proceedings of the Sixth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '98)*, Montreal, Canada (July 1998).
- [10] Frumkin, M., Wijngaart, Van Der. "NAS Grid Benchmarks: A Tool for Grid Space Exploration." *Proceedings of SC2001 Conference on High Performance Networking and Computing, Denver, CO, IEEE Computer Society Press* (Nov., 2001).
- [11] Sun, Y., Wang, J. and Xu, Z. "Architectural Implications of the NAS MG and FT Parallel Benchmarks." *1997 Advances in Parallel and Distributed Computing Conference (APDC '97)*, National Center for Intelligent Computing Systems(NCIC) Chinese Academy of Sciences, Shanghai, China (March 19 - 21, 1997).
- [12] Yoo, A. Jette, M. "The Characteristics of Workload on ASCI Blue-Pacific at Lawrence Livermore National Laboratory." *1st International Symposium on Cluster Computing and the Grid*, Brisbane, Australia (May 15 - 18, 2001).
- [13] Almojel, A. "An Architecture-Independent Workload Characterization Model For Parallel Computer Architectures." a *Doctoral Dissertation* and was published by NASA Goddard Space Flight Center, Dissertation Series, CESDIS TR-97-204 (June 1997).
- [14] <http://www.nas.nasa.gov/NAS/NPB>.
- [15] Wong, F., Martin, R., Arpaci-Dusseau, R. and Culler, D. "Architectural Requirements and Scalability of the NAS Parallel Benchmarks." *Proceeding of SC99 Conference on High Performance Networking and Computing, IEEE Computer Society Press, Portland, OR.* (Nov. 1999).
- [16] Cantonnet, F. "UPC Performance and Potential: A NPB Experimental Study." *Proceeding of SC2002 Conference on High Performance Networking and Computing, Baltimore, MD., IEEE Computer Society Press* (Nov. 2002).

**ILP**

( / / / / )

( ) ( )

.( )

( )

