University of Delaware
Department of Electrical and Computer Engineering
Computer Architecture and Parallel Systems Laboratory

P3I
The Delaware Programmability, Productivity and Proficiency Inquiry

Joseph B. Manzano *Yuan Zhang* *Guang R. Gao*
Department of Electrical & Computer Engineering, University of Delaware
Newark, Delaware 19716, U.S.A
{jmanzano,zhangy,ggao}@capsl.udel.edu

CAPSL Technical Memo 61
15 April 2005

Abstract

New advancements on high-productivity computing systems have shown the weaknesses of existing parallel programming models and languages. To address such weaknesses, a number of researchers have proposed new parallel programming models and powerful programming language features that can meet the challenges of the emerging HPC Systems. However, the success or the failure of a new programming model and accompanied language features need to be evaluated in the context of their productivity impacts. In this paper, we report a productivity study that was conducted at the University of Delaware in the context of the IBM PERCS project. Such project is being funded via the DARPA/HPCS enterprise. In particular, our study is centered on the productivity impact of a new and key programming construct called Atomic Sections that is jointly proposed by our group and our colleagues at IBM.

1 Introduction

New advancements on High Productivity Computing (HPC) Systems have shown the weaknesses of existing parallel programming models and languages. To address such issues, a number of researchers have proposed new programming models and powerful programming language features that can meet the challenges of the emerging HPC Systems. However, the success or failure of such programming models and accompanied language features needs to be evaluated in the context of their productivity impacts. A small overview of the new approach to language design is depicted in Figure ??. A key source of complexity (and a productivity deterrent)

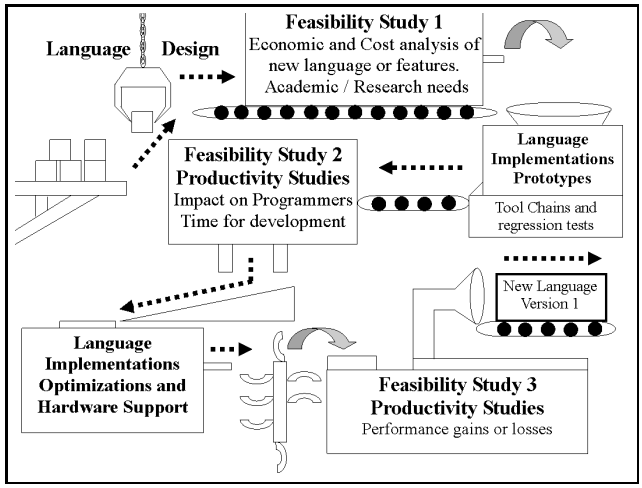

Figure 1: A New Approach to Language Design

in parallel programming arises from fine grained synchronization which appears in the form of lock / unlock operations or critical sections. An effect of these constructs in productivity is to put excessive resource management on the programmer. Thanks to this, the probability that a programmer's errors appear increases. Another side effect is the hidden overhead of the underlying synchronization actions and their accompanying data consistency operations. This

overhead reduces the amount of scalable parallelism that could have been achieved. Atomic Sections have been proposed as a parallel programming construct that can simplify the use of fine grained synchronization, while delivering scalable parallelism by using a weak memory consistency model. This construct has been implemented under OpenMP_XN, an extension of OpenMP in the context of PERCS; that was implemented by the authors of this paper. Based on this prototype implementation, a productivity study (The Delaware Programmability, Productivity and Proficiency Inquiry or P3I) was designed and implemented. The main focus of P3I is to test the productivity of Atomic Sections overall and the productivity of them in each phase of an application development (designing, parallelization and debugging). P3I, in its first implementation, was conceived not for measuring performance, but to measure programmability and debug-ability given a set of short programming exercises. Another feature that makes this study unique is its weighting factor, which is a function that will change the amount of participation (time) of each participant, according to its expertise in the HPC domain. An overview of P3I weights and weighting scheme is given in the subsequent sections. The purpose of this paper is to provide an outline of P3I, its framework and methodology. Section 2 provides a brief overview of the new constructs: Atomic Sections. Sections 3 and 4 present a high level overview of P3I and a deeper view of its internals, data collection methods, and parts, respectively. Section 5 explains the weighting procedures and its overall purpose. Sections 6 and 7 present results and conclusions based on the first implementation of P3I. Finally, section 8 presents related and future work.

2 Atomic Sections: Overview

An Atomic Section is a construct that was jointly proposed by the authors of this paper and their colleagues at IBM. This construct has two implementations. One of them is under IBM's X10 language [**?**]. It is used for local synchronization on IBM's PERCS. Atomic Sections' second implementation is under the OpenMP_XN programming model. OpenMP[1] is the standard programming model for Shared Memory Processors (SMP) machines. Due to its status and the plethora of resources available for it, OpenMP became an excellent platform to test new parallel programming constructs. The Delaware group, which is comprised by the authors of this paper, took the OpenMP and extended it with Atomic Sections. Hence, the new programming model became OpenMP with extensions or OpenMP_XN for short. From now on, every time that the term Atomic Sections is used, the implementation under OpenMP_XN should be assumed, unless stated otherwise. An Atomic Section is defined as a section of code that is intended to be executed atomically, and be mutually exclusive from other conflicting atomic operations. The word "conflicting" in the previous statement is one of the main reasons that Atomic Sections is different from other synchronization constructs in OpenMP. Atomic Sections that are conflicting should be guarded against each other but they shouldn't interfere with the ones

[1]OpenMP is a parallel programming extension for C/C++ and FORTRAN. It uses the fork and join model as its parallel programming model [**?**]

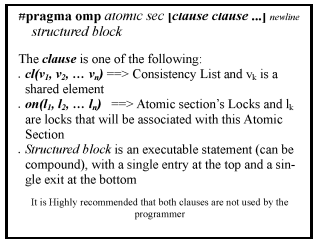

Figure 2: Atomic Section's Syntax

```
#pragma omp parallel for private(b)
for(i = 0 ; i < 10; ++i)
{
  b = rand()%255;
    #pragma omp atomic_sec
    {
    a = b * c;
  }
}
```

Figure 3: Atomic Section's Example

that do not conflict with them. Standard OpenMP[2] offers two extreme cases when it comes to interference and synchronization. Its critical section construct provides a global lock that ensures that only a critical section is running and the others are waiting, regardless of the effects of their execution. On the other hand, OpenMP's lock functions put the responsibility on the programmer to detect if the code interferes (data race) with other protected blocks of code, and lock it according to his/her judgment. Atomic Sections provide the middle ground in which the programmer is free of lock management and the compiler will take care to run the sections in a non-interfering manner. The syntax and a code example of Atomic Sections are presented in figures **??** and **??**. Each Atomic Section's instance is associated with a structured block of code. This block is defined as a piece of code that has only one entry point and one exit point. This a brief introduction to this construct, for a more in depth explanation please refer to [**?**]. During the conception of this construct, another aspect became apparent: how will programmers react to this construct? Therefore, the Delaware group developed P3I to have an idea about the construct impact on programmers and to create a methodology / framework template for future studies.

3 P3I: Overview

The main objective of P3I is to measure the productivity impact that a new construct has. From this main objective, many small questions arise. How much impact will the new construct have on the programmer? How will the construct affect each phase of application development? Other questions about the study itself are raised. Is there any way to ensure a correct distribution of our sample, or to pre-filter the results such that the data that we considered more relevant (i.e. novice and average programmers) have more weights than others? This productivity study tries to answer these questions and provides a solid platform for future work and iterations. P3I had a total of fifteen participants. These participants came from a pool of graduate and undergraduate students in the department of Electrical and Computer Engineering at the University of Delaware. The participants attended two meetings in which the study was presented to them. The study itself is accessible through a group of web pages call the Web Hub. In the Web Hub, the participants can review, download and fill the different parts of the

[2]OpenMP also offers the atomic directive but this construct is extremely restricted. It is limited for read modify write cycles and for a simple group of operations

study. The Web Hub is divided into phases, which each participant should take in order. The first phase is dubbed Phase 0 and it consists of a Web Survey, Web log and a programming exercise. All the other phases of the study contain only a Web log and programming exercise. The purpose of Phase 0 is to get the participants familiar with the study form and infrastructure. After Phase 0, there are three more phases which represent the core of the study. For a complete description of all phases and the data collection infrastructure, refer to the next section. A code excerpt from Phase 0 is presented in figure **??**. This code excerpt presents the code involving a bank transaction using lock / unlock OpenMP construct and Atomic Section construct. The participants use a modified version of a free source OpenMP compiler, called Omni [**?**], which has been modified to support OpenMP_XN. This study's main metric

```
omp_init_lock(&(tmp-
>lock));
...
void deposit(long int
depo, int ID)
{
   struct customers *tmp;
   tmp = found(ID);
   omp_set_lock(&(tmp-
>lock));
   {
      tmp =
      tmp->balance += depo;
   }
   omp_unset_lock(&(tmp-
>lock));
}
...
omp_destroy_lock(&(tmp-
>lock));
...
```

```
void deposit(long int
depo, int ID)
{
  struct customers *tmp;
  #pragma omp atomic_sec
  {
   tmp = found(ID);
   tmp->balance += depo;
  }
}
```

Figure 4: Code Excerpt from Phase 0

is the time to correct solution, which is calculated thanks to the data collection infrastructure explained in section 4. As stated before, this study is not designed to measure performance (even though that will be one of its objectives in future iterations). The Web Survey is used to calculate the weights that will be used in this study. Its structure, purpose and results will be presented in sections 5 and 6. The programming exercises in each of the phases are designed to put programmers in different contexts that appear in the HPC domains. These situations can be described as developing a complete parallel application, debugging a parallel code that deadlocks and parallelizing a serial application. Finally, all the results will be filtered with the weights such that the final data will be affected by the level of expertise of the participants.

4 P3I Infrastructure and Procedures

The main study is composed of four parts or phases. The main webpage can be found at [**?**]. In there, the participants can learn about the study itself, its main metric, and each phase's time limit. Moreover, extra materials can be accesses through this website. These extras include tutorials about OpenMP, POSIX threads, a brief explanation about Atomic Sections,

the OpenMP_XN compiler, and instructions on how to install it. Figure ?? provides a picture depicting the structure of the Web Hub. The first phase is dubbed Phase 0: The Unfortunate

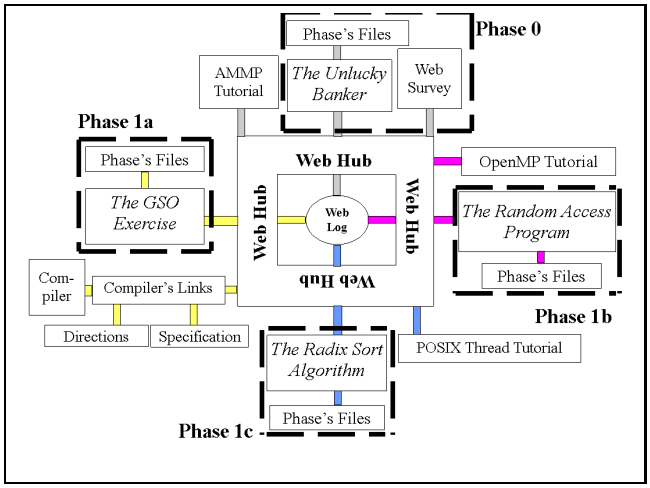

Figure 5: P3I Infrastructure

Banker and its main objective is to show the procedures for running all the subsequent phases. The phase itself is subdivided into a Web Survey, a Web Log and the programming exercise: The Banker. The Web Survey is the first step of the study and it is comprised of 24 questions that test the participant's knowledge in parallel programming extensions, parallel execution models and hardware support for parallel models. More about the survey score system and its importance will be discussed in the next section. The Web Log is designed to capture subjective data about the starting and stopping time of each phase. It also has a special section for participants' comments and answers to some questions raised by each phase. The questions are about aspects of the programming exercises and their answers are simple but require a certain understanding about the exercise. Most of these questions are optional. Every phase requires the participant to record data in the Web Log at the beginning and at the ending of each phase. The programming exercise consists of three files. The first file is a make file that will be used to build the source files. The next file is a bash script which will collect time stamps, user info, compiler information and program output. The participants are required to use this script to run their application. Finally, a skeleton C source file is provided for the participants to use. In Phase 0, this programming exercise is a simple simulation of bank transactions between four branches scattered across the country. Each branch is simulated as a thread that receives transactions and synchronizes them with locks or Atomic Sections. There is a different group of files for each required construct. Therefore, the participant will run this phase twice; one for locks and one for Atomic Sections. In this phase, the source files are complete running applications that were ready to run "out of the box". All the phases have the same structure as this phase. Therefore, this phase can be seen as an "acclimation" phase in which the participants learn how to run the phases and its parts. Phase 1 is the core of the study and consists of 3 sub-phases and several exercises. The first sub phase is called Phase 1a: the GSO exercise. This exercise will present a hypothetical case in which a

Gram Schmidt Ortho-normalization is required to create an ortho-normal basis. The exercise should be completed from scratch and the final code should be parallelized with OpenMP_XN C function calls and pragmas. The participants are only given helper functions for them to use in their code. One of the requirements for a successful run is that a provided check function returns true when testing the basis. Another requirement for completion is that two versions are created (one for locks and one for Atomic Sections) and that each version successfully runs. Some extra information is provided in the webpage of the phase [**?**]. This specific exercise was developed to test the programmer's abilities in designing, parallelizing and debugging an application. Phase 1b is dubbed The Random Access Program. It is based on the Random Access exercise which is used to test memory bandwidth systems. The program contains a huge table that is randomly accessed and updated. At the end of the execution the reversed process is applied to the same table and the table is checked for consistency. If the number of errors surpasses one percent, the test has failed. The synchronization in this case applies to each random access of the elements in the table. Intuitively, the number of errors that might appear in the final table reduces considerably when the table is made large enough. This makes the use of synchronization constructs useless for the program, and it is actually one of the questions that is asked in the webpage of this phase [**?**]. In this phase, the subjects are given a serial version of the program and are asked to parallelize it with OpenMP_XN. As before, two versions are required to complete this phase (locks and Atomic Sections). An extra version can be completed and it consists of the program without any synchronization constructs. This exercise simulates the scenario in which programmers need to change serial codes to parallel implementations. Phase 1c is called The Radix Sort Algorithm and is an implementation of this famous algorithm. The algorithm itself is explained in this sub phase's webpage [**?**]. The participants are given a buggy parallel implementation of this algorithm. There are three bugs in the algorithm that relate to general programming, parallelization and deadlocks. All three bugs are highly dependent and when one is found and solved, the others become apparent. As before, a lock and an Atomic Section version are required. The extra questions in this section involve the identification of the bugs, why it becomes a problem and possible solutions. The main objective of this section is to measure the debug-ability of a parallel code that involves synchronization constructs. A summary of the Methodology and Framework is given by Figure **??**. All data collected from the phases is saved to a file that is in the possession of the organizers of the study. The data that is collected is ensured to be private and it is only made available to one of the organizers of the study. The identity of the participants and their results are kept secret so no possible repercussion of their participation can arise. This process is a "double blinded" process since the participants cannot access their results and the rest of the Delaware group doesn't know who participated or for how long they stayed in the study.

5 The Web Survey: Purpose

The web survey is the first part of the P3I and it is mandatory for all participants. It consists of 24 questions that range from a simple "With which programming language are you familiar?"

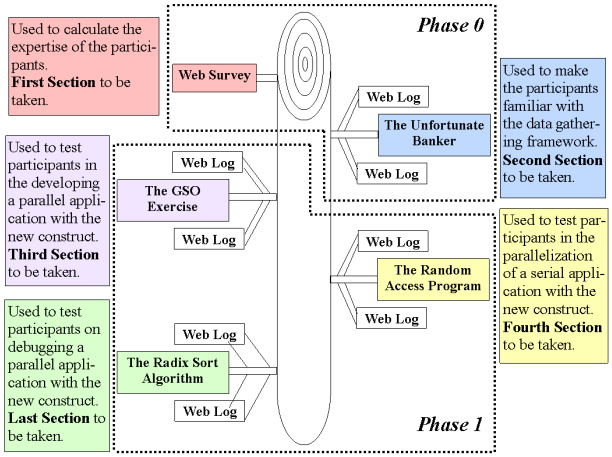

Figure 6: P3I Infrastructure and Methodology

to more complicated questions such as "How much you know about the fork and join parallel programming model?" In the web survey, participants will check boxes or radial buttons to decide their level of expertise in a range of 1 (least expert / Never heard about it) to 5 (expert / Use frequently). Each question has a maximum score of 5 - except for the first one that has a value of 6 - and some questions are left out of the final computation since they deal with hardware support. An expert score in the web survey is 106. When a participant finishes the web survey, his/her score is calculated. Afterward, a ratio is taken with respect with the expert score. This will be called the expertise level percentage. All these calculations are called "Weight Calculations" and they will be kept intact in future iterations. Finally, the expertise level is subtracted from one to produce the participant's weight. This weight will be used to filter the data by multiplying it with each participant's time. This process will amplify the contribution of less expert programmers to the study. These final steps are named "The Weighting Function" and it will be modified in future iterations of the study. That being said, P3I - in its first iteration - target low level and average programmers. It also has the ability to "weed" out all high expertise participants. This will prevent skewing of data from the high expertise end, but it will amplify on the low end. This scheme can be considered a "Low Expertise Weighting Function". Two other schemes have been considered, and they will be applied in the next iterations of P3I. For the explanation of these future schemes, please refer to section 8.

6 P3I Results

The results of the study consist of the average time of all participants in each sub-phase and sub-section. Each time data is weighted with the participant's weight before the average is calculated. Each participant has to run the experiments in a Sun SMP machine with 4 processors and a modified version of the Omni Compiler. The results for the weights are presented in figures ?? and ??. It shows that the distribution of expertise among the participants approaches

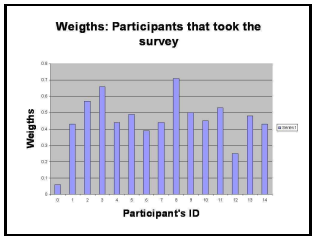

Figure 7: Weight of each Participant

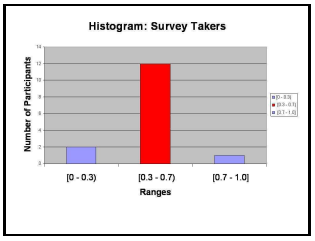

Figure 8: Histogram of the Weights

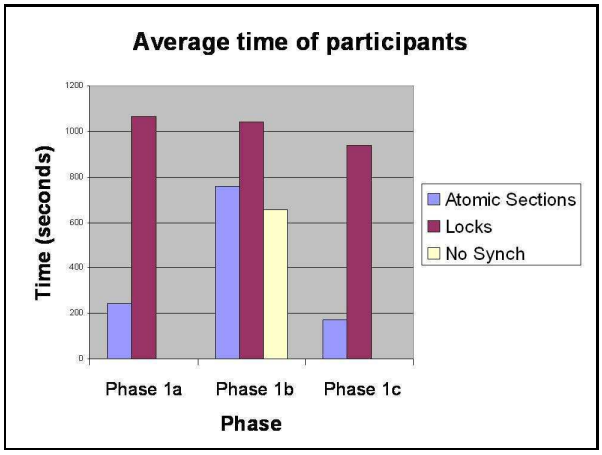

Figure 9: Weighted Average Time Data for Each Phase

a normal distribution. This result can be used in future iterations to break the population into samples. This will also allow researchers to test several hypotheses about productivity. More about these future schemes will be presented in section 8. Figure **??** provides the final results that have been modified by the weight's data. A complete discussion about the results and the weights of the participants are given in the next section.

7 Analysis and Conclusions

As shown by tables 1a and 1b, the weights in this study formed a slightly skewed normal distribution. This will ensure that most of the results will be weighted against a close range of values. Moreover, the actual weights that were presented in the study are in the range of 0.42 to 0.55. This means that the data for this study was not affected much by the weights. Also, this means that the population is not suitable for sampling since all the groups has the same level of expertise. As Table 2 shows, there is a considerable reduction of time to solution in all the phases. However, the small sample space hinders this study from making a stronger case. Overall, there is a dramatic reduction of the time to correct solution, and each phase also shows this reduction. In Phases 1a and 1c, this reduction is in factors of five. In this data, the sequencing information should be considered. This information is obtained by recording the order in which the sub sections were taken within each sub phase. This is important because

there is always a learning period in which the participants get familiar with the code. Even in these cases, the reduction of time to solution is also present. Moreover, this study can be augmented and redesigned thanks to data gained from this study and many others. Therefore, the first iteration of P3I serves as a solid foundation for more iterations of this study or others.

8 Future and Related Work

Based on several new studies and interactions with other productivity groups, the Delaware group already formed a base for the next iteration of P3I. The first improvement will be to greatly reduce the human factor in data collection and extend the data collection infrastructure for both objective and subjective data. Currently, the data is subject to human interaction. To reduce this effect, a modified shell and an instrumented compiler will be used in the next iteration. The shell will collect all data concerning the user activity on it. Moreover, the shell will also collect editor history. The compiler will silently collect warnings and errors on the application being compiled. The web log should be enhanced with more parts as an estimate on compilation, debugging and designing. More relevance should be given to comments. The human factors and behavior should be considered more. A larger population should be considered and sampling of the population should take place. Samples should eliminate the sequencing problem by creating control and experimental groups. A broader area should be considered, as computer scientists must be included in the sample. The web survey should be extended to include questions about compiler specifics and multithreaded knowledge. Versions of POSIX threads, MPI exercises and OpenMP programs should be created as a familiarization exercise for novice programming (i.e. extending Phase 0). Finally, a new incentive system should be instituted (i.e. exercises can be homework or projects in college-level multithreaded classes). These suggestions come from work done on the University of Maryland at the helm of Vic Basili [**?**] [**?**]. Special thanks to Vivek Sarkar, Kemal Eboglicu, Vic Basili and all the Delaware group's collaborators from IBM Watson Research Center for all their help on bringing this study to completion.

9 Acknowledgment

This work has been supported in part by the Defense Advanced Research Projects Agency (DARPA) under contract No. NBCH30390004.

References

[1] Omni openmp compiler project. `http://phase.hpcc.jp/Omni/home.html`. Parallel and High Performance Applicational Software Exchange (PHASE).

[2] Openmp: Simple, portable, scalable smp programming. `http://www.openmp.org/drupal`. OpenMP Architecture Review Board.

[3] S. Asgari, V. Basili, J. Carver, L. Hochstein, J. K. Hollingsworth, F. Shull, and M. Zelkowitz. Challenges in measuring hpcs learner productivity in an age of ubiquitous computing. In *Proceeding of Workshop on Software Engineering and High Performance Computing Applications (held at ICSE)*, 2004.

[4] J. Carver, S. Asgari, V. Basili, L.Hochstein, J. K. Hollingsworth, F. Shull, and M. Zelkowitz. Studying code development for high performance computing: The hpcs program. In *Proceeding of Workshop on Software Engineering and High Performance Computing Applications (held at ICSE)*, 2004.

[5] K. Ebcioglu, V. Saraswat, and V. Sarkar. X10: an experimental language for high productivity programming of scalable systems. In *Proceeding of the Second Workshop on Productivity and Performance in High End Computers*, pages 45 – 51, February 2005.

[6] J. B. Manzano, Y. Zhang, and G. Gao. Productivity study. `http://www.capsl.udel.edu/courses/eleg652/2004/productive`, January 2005. Computer Architecture and Parallel System Laboratory (CAPSL).

[7] J.B. Manzano, Y. Zhang, and G. Gao. Phase 1a: The dot product exercise. `http://www.capsl.udel.edu/courses/eleg652/2004/productive/dotp.htm`, January 2005. Computer Architecture and Parallel System Laboratory.

[8] J.B. Manzano, Y. Zhang, and G. Gao. Phase 1b: The global updates per second benchmarks. the random access program. `http://www.capsl.udel.edu/courses/eleg652/2004/productive/gups.htm`, January 2005. Computer Architecture and Parallel System Laboratory.

[9] J.B. Manzano, Y. Zhang, and G. Gao. Phase 1c: The radix sort algorithm. `http://www.capsl.udel.edu/courses/eleg652/2004/productive/rsort.htm`, January 2005. Computer Architecture and Parallel System Laboratory.

[10] Y. Zhang, J.B. Manzano, and G. Gao. Atomic section: Concept and implementation. In *Mid-Atlantic Student Workshop on Programming Languages and Systems (MASPLAS)*, 2005.