LA-UR 9 - 3213

Approved for public release;
distribution is unlimited

CONF-980212--

Title:

A Lower Bound for Quantifying Overlap Effects:
An Empirical Approach

Author(s):

Federico Bassetti

RECEIVED

DEC 01 1997

OSTI

19980401 035

Submitted to:

1998 IEEE Int'l. Performance, Computing, and
Communications Conference
February 16-18, 1998
Phoenix/Tempe, Arizona

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

Los Alamos
National Laboratory

MASTER

Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the University of California for the U.S. Department of Energy under contract W-7405-ENG-36. By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. The Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

DTIC QUALITY INSPECTED 3

Form 836 (10/96)

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

A Lower Bound for Quantifying Overlap Effects: An Empirical Approach

Federico Bassetti
fede@lanl.gov
Los Alamos National Laboratory
Mail Stop B287
Los Alamos, NM 87545

Abstract

Among the many features that are implemented in today's microprocessors there are some that have the capability of reducing the execution time via overlapping of different operations. Overlapping of instructions with other instructions, and overlapping of computation with memory activities are the main way in which execution time is reduced. In this paper we will introduce a notion of overlap and its definition, and a few different ways to capture its effects. We will characterize some of the ASCI benchmarks using the overlap and some other quantities related to it. Also, we will present a characterization of the overlap effects using a lower bound derived empirically from measured data. We will conclude by using the lower bound to estimate other components of the overall execution time.

1. Introduction

In the years 1994-95 superscalar processors, with three or more issues per clock, made their appearance on the market. All the major vendors were, and still are, supporting superscalar processors such as: MIPS R10000, Intel Pentium Pro, IBM Power2, HP 8000, DEC Alpha, Sun UltraSPARC. Among the many features that are implemented in today's microprocessors there are some that have the capability of reducing the execution time via overlapping of different operations. Overlapping of instructions with other instructions, and overlapping of computation with memory activities are the main ways in which execution time is reduced. For all those processors we can also talk of overlap between memory accesses at different levels of the memory hierarchy. In order to exploit even more parallelism between activities, some processors support out of order execution, and not blocking caches with more than one outstanding miss. In the study presented the MIPS R10000 has been chosen as reference for the experiments. The results presented in this paper are focused on the estimation of overlap effects. A lower bound for the overlap is computed using an empirical approach based on measured data. The result is then applied to compute bounds to quantify other effects. The results presented in the paper are obtained using the ASCI¹ benchmarks. Due to the novelty of both the ideas and their implementation in today's processors no other study, to the best of our knowledge, has approached the problem of quantifying the effects of overlap.

2. Overlap

The analysis in the following sections uses a simplified mean value parameterization [1] to separate CPU execution time from stall time due to memory loads and stores. Figure 1 is a pictorial description of modeled times.

¹ Department of Energy Accelerated Strategic Computing Initiative

Figure 1. Relationship of modeled times

The model projects the overall *cpi* of an application as a function of CPU execution time and average memory access times:

$$(1) \quad cpi = cpi_0 + \sum_{i=2}^{memory} h_i * t_i$$

where cpi_0 is defined to be the *cpi* of the application assuming that all memory accesses are from an infinite L1 cache and take *one* clock period (i.e., the $i=1$ term is included in cpi_0), and h_i and t_i are, correspondingly, the hits per instruction and average non-overlapped access times for the i th level in the memory hierarchy. The second term in the right hand side of equation (1) is also referred to as cpi_{stall} . If no overlap of CPU execution and memory accesses occur, every memory access to the i th level incurs the full round-trip latency, which we denote as T_i . We define a measure of the overlap of memory accesses with computation as m_0 in (3.a), where *cpi* is formulated as in (2). The overlap contribution expressed in clocks per instruction is given in (3.b).

$$(2) \quad cpi = cpi_0 + (1 - m_0) \sum_{i=2}^{memory} h_i * T_i$$

From (1), m_0 is derived as one minus the ratio of the average memory access time to the maximum memory access time.

$$(3.a) \quad m_0 = 1 - \frac{\sum_{i=2}^{memory} h_i * t_i}{\sum_{i=2}^{memory} h_i * T_i} \quad (3.b) \quad cpi_{overlap} = m_0 \sum_{i=2}^{memory} h_i * T_i$$

In [9] can be found a preliminary study on understanding what are the features that have an impact on overlap, and in what measure they contribute to the whole overlap. The approach used in [9] makes use of a combination of measurements, and least square fitting.

3. ASCI Benchmarks

Four applications, which form the building blocks for many ASCI simulations, were used in this study. A more detailed characterization of the ASCI benchmarks is presented in [2,3].

3.1. Code Descriptions

SWEEP3D is a three dimensional solver for the time independent, neutral particle transport equation on an orthogonal mesh [4]. The first-order form of the transport equation is solved by sweeping through the spatial mesh along discrete directions (ordinates). In SWEEP3D, the main part of the computation consists of a "balance" loop in which particle flux out of a cell in three Cartesian directions is updated based on the fluxes into that cell and on other quantities such as local sources, cross section data, and geometric factors. The cell-to-cell flux dependence implies a recursive or wavefront structure, i.e. a given cell cannot be computed until all of its upstream neighbors have been computed. The specific version used in these tests was a scalar-optimized "line-sweep" that involves separately nested, quadrant, angle, and spatial-dimension loops. In contrast with vectorized plane-sweep versions of SWEEP3D, there are no gather/scatter operations and memory traffic is significantly reduced through "scalarization" of some array quantities. Because of these features, L1 cache reuse on SWEEP3D is fairly high (the hit rate is about 85%). A problem size of N implies N^3 grid points.

Figure 2. Characterization of the ASCI benchmarks on a SGI Origin 2000 system.

HYDRO is a two-dimensional explicit Lagrangian hydrodynamics code based on an algorithm by W. D. Schulz [5]. HYDRO is representative of a large class of codes in use at the Laboratory. The code is 100% vectorizable. An important characteristic of the code is that most arrays are accessed with a stride equal to the length of one dimension of the grid. HYDRO-T is a version of HYDRO in which most of the arrays have been transposed so that access is now largely unit-stride. A problem size of N implies N^2 grid points.

HEAT solves the implicit diffusion PDE using a conjugate gradient solver for a single timestep. The code was written originally for the CRAY T3D using SHMEM. The key aspect of HEAT is that its grid structure and data access methods are designed to support one type of adaptive mesh refinement (AMR) mechanism, although the benchmark code as supplied does not currently

handle anything other than a single-level AMR grid (i.e. the coarse, regular level-1 grid only). A problem size of N implies N^3 grid points.

NEUT is a Monte-Carlo particle transport code. It solves the same problem as SWEEP3D but uses a statistical solution of the transport equation. Particles are individually tracked through a three dimensional mesh where they have some probability of colliding with cell material. The output from the particle tracking is a spatial flux discretized over the mesh. Vector (or data parallel) versions of this type of code exist which track particle ensembles rather than individual ones. A problem size of N implies N^3 grid points and 10 particles per grid point.

3.2. Performance Characteristics

In this section we present some single-processor characteristics of the benchmark codes as obtained from performance counters on the Origin 2000 [6]. Detailed performance characteristic data for these codes were collected on a 4-MB L2 Origin 2000 system [7].

Mem/FLOPS is the ratio of memory references to floating point instructions and reflects the density of *load/store* instructions in a code. The results show the number of accesses is related to *FLOPS* by a small constant (greater than one) and the growth rate of both memory accesses and *FLOPS* is $O(n)$. In HEAT, the high *Mem/FLOPS* ratio is due to *gather/scatter* memory accesses in the code. The codes' overall *cpi* curves are generally the inverse of their corresponding *MFLOPS* curves; that is, an increasing *cpi* corresponds to a decreasing *MFLOPS* at nearly the same slope and vice versa. The *cpi* of three of the codes (HEAT, HYDRO and SWEEP) is strongly dependent on problem size. Although not shown in the figures, we calculated TLB hit ratio and branch prediction hit ratio. The calculation shows that MIPS R10000 processor uses a good technique for performing speculative branch prediction. All four benchmark codes (HEAT, HYDRO, HYDRO-T and SWEEP) have branch prediction hit ratios over 99%. This means that over 99% of speculated branch predictions are taken in real executions. TLB hit ratios for all these codes are higher than 98%.

4. A Lower Bound for Estimating Overlapping Effects

A formal definition of lower bound can be found in [8]. Under the assumption that we will present in this section, a lower bound to estimate the effects of overlap can be computed. A lower bound for each code, and for each different problem size will be obtained. The lower bound can be seen as a portion of the actual overlap contribution for a particular code and problem size on a given architecture. This portion is less than the actual overlap, according to the definition of lower bound. In this paper we cannot claim that this lower bound is sharp, but that it is tight enough. The terminology and the concepts adopted to derive the lower bound are the same as those used in [9], and presented previously in section 2. Equation (2) was derived by transforming (1), and incorporating the overlap in the stall component. In particular, $1-m_o$ estimates the fraction of memory references that pay full latency (indicated with the presence of T). If now we consider (2) in a hypothetical scenario with no overlap, then the *stall* component in (2) will be maximized.

In fact, considering equation (2) with the maximized stall component produces a different cpi_0 contribution, as formulated in (4).

$$(4) \quad cpi = cpi_0 + cpi_{stall}^{max}$$

Equations (2) and (4) are the same in their right hand sides. Equation (4) differs from (2) in the way the two components in the right hand side distribute their values, because of the maximized cpi_{stall} part. We can say that cpi_{stall} component is maximized since it is actually representing the worst case, in which every memory access sees, from the processor perspective, full latency. As a consequence of the fact that the right hand side in (4) must add up to cpi , the value for cpi_0 is then the minimum possible². At this point we can algebraically derive the minimum cpi_0 as described in (5).

$$(5) \quad cpi - \sum_{i=2}^{memory} h_i * T_i = cpi_0 = cpi_0^{min}$$

In figure 3 we show the components of equality (5) using the ASCI benchmarks [2,3] as test codes on the currently available SGI Origin 2000 system [7]. The results presented are based on measured data obtained using the hardware performance counters available on the MIPS R10000 chip [6]. Counts for number of cycles and number of instructions executed by a given code, are used to compute the overall cpi . The number of misses at each level of cache enables us to compute the hit ratios h_i , since we can measure the instructions executed. The *maximized* cpi_{stall} can be computed since the T_i 's are given by the manufacturer [7,11]. The minimum cpi_0 is then computed by difference.

One can see that the maximized cpi_{stall} could be greater than the overall cpi , producing a negative minimized cpi_0 . Since the overall cpi is fixed and known, the component that will maintain the balance in (4) is the cpi_0 . Thus, the fact that the maximized cpi_{stall} is greater than the overall cpi tells us that overlap effects are present in the code analyzed. With this knowledge a first estimation of overlap impacts is possible. Figure 3 shows the difference between the overall cpi and the maximized cpi_{stall} as first quantification of the overlap on an Origin 2000 system using the ASCI benchmarks.

In our assumptions we maximized the stall component and minimized as a consequence the cpi_0 component. The cpi_0 is setting the minimum computation contribution to the overall cpi . In order to reach our lower bound of overlapping effects, we have to add a concept specific of the architecture where the measurements are taken. The component needed is the best cpi_0 for the architecture under consideration. The best cpi_0 will coincide also with the best overall cpi , because of the way in which cpi_0 has been defined. The best cpi_0 for the architecture under study

² Given a code and a problem size the quantity represented by cpi_0 is the least computation contribution needed, under the assumption of maximum stall time, in order to get the same cpi value as in (2).

is equal to 0.25 [10] since the MIPS R10000 microprocessor is capable of graduating four instructions within the same cycle.

Figure 3. Minimum cpi_0 computed with data obtained for the ASCI benchmarks on Origin 2000 systems.

Given the overlap as expressed in (3.b), a lower bound is defined in (6).

$$(6) \quad cpi_{overlap}^{\min} = \begin{cases} 0.25 - cpi_0^{\min} & \text{if } cpi_0^{\min} < 0.25 \\ 0 & \text{otherwise} \end{cases}$$

From the second case of (6) we can only say that the minimum overlap contribution is null, but that doesn't imply that the actual overlap contribution is null. From the first case of (6) we can say something about those cases where we know that overlap is present. This part is in fact saying that the overlap contribution cannot be less than that value. In figure 4 we show the lower bound for the overlap, expressed in cycles per instruction, together with the overall cpi . From the chart one can see how each given code benefits from an overlap that is at least $cpi_{overlap}$ to obtain the overall cpi shown. With the data in figure 4 it is possible to relate the meaning of (6) to some codes. From the chart one can see that there are cases in which the overlap is null. There are two possible interpretations to this case: the code under study doesn't have potential for overlap (ex. every instruction is dependent from the previous); or, the code under study is cache resident, and doesn't have a significant stall component. All the cases for the *NEUT* benchmark are an instance of the latter scenario; their working set fits in cache. From figure 3 one can see that for these code even the maximized cpi_{stall} is significantly small, and the hit ratio for L1 is virtually 100% (figure 2). Thus in this context a null lower bound for the overlap indicates a code that performs well, and that needs no significant overlap contribution to achieve good performance.

Figure 4. The lower bound for the overlap effects using the ASCI benchmarks on Origin 2000 systems.

5. Using the Lower Bound

There are different ways in which we could use the lower bound:

- We can use it to characterize the overlapping capabilities of two different codes on the same architecture.
- We can use it to characterize the overlapping capabilities of a code with varying problem size.
- We could use it to characterize the same code over different architectures.
- We could use it to estimate the capabilities of compilers in generating low-level code that takes advantage of overlapping features of the architecture.

The reasons that brought us to undertake this study are related to a larger project in which performance modeling has a key role. We used the lower bound to compare the overlapping capabilities of two different systems currently available to us: a Power Challenge system and an Origin 2000 system, both from SGI.

Figure 5. Lower bound for the effects of overlap. Results for ASCI codes on a Power-Challenge system.

The lower bound effects on a Power Challenge system are presented in figure 5. In particular, during our studies we wanted to quantify the impact on performance of two features that are different on the two systems and that, according to [9], can be seen as overlap effects. These features are number of outstanding misses, and main memory latency. In the Origin 2000 systems the number of outstanding misses is increased to a value of 4 compared to 1.5 on the Power Challenge. The latency to main memory of the Origin 2000 systems has been reduced to 80 cycles against the 205 cycles of the Power Challenge. Figure 6 shows the characterization of the ASCI benchmarks on the two different systems using the lower bound for the overlap. From the chart one can see that more overlap resulted on the Power Challenge system. From the data in figures 5 and 3 it is clear that the Origin system is performing better. The effect of the overlap can be interpreted in two ways: high overlap is a consequence of a code that is using compiling and architecture features at their best, and that the nature of the code is such that those feature are needed. High overlap is a consequence of poor memory behavior; frequent and long memory access create a good potential for overlap, but this does not necessarily imply better performance than a system with a smaller overlap. The data from figure 6 present an instance of the latter scenario. The Origin system is a better machine from an architecture point of view; its stall time for main memory is significantly smaller, and the frequency of main memory access is reduced. Thus on Origin the codes spend less time stalling. Also, the implementation of the codes doesn't take full advantage of the higher number of outstanding misses available on the Origin. A more detailed comparison of the two systems using the ASCI codes can be found in [2,9].

Figure 6. A comparison between the overlap effects on two different systems using the lower bound as a metric. PC is a power challenge system with 1Mb of L2, 2 outstanding misses support, and 205 cycles of latency to main memory. O2k is an Origin 2000 system with 4Mb of L2, 4 outstanding misses support, and 80 cycles of latency to main memory.

An immediate consequence of our study is the possibility of characterizing the impact of different features on the overlap, and as well as how the overlap effects may vary when some of the features are changed. In the following section we will use the lower bound result to bound the estimation of cpi_{stall} and cpi_0 .

6. Derived Bounds for cpi_{stall} and cpi_0

A use of the lower bound result enabled us to bound some terms of the equality (1). In fact, not all the terms in (1) can be directly measured. The effective latencies t_i and cpi_0 are not directly measurable. We can use the lower bound for the overlap to compute a bound for both the unknowns.

Upper bound for cpi_{stall}

Before we approach the problem, it is necessary to mention the result from [2,9] which shows that cpi_0 can be considered as a constant for a given problem. This means that each different code will have its own cpi_0 , but this value converges to the same value when the problem size increases. Also, cpi is a good approximation of cpi_0 if the problem size is small enough to make the problem fit in primary cache [2,9]. We can now rewrite (1) where we will replace cpi_0 with cpi_{L1} and obtain (7).

$$(7) \quad cpi \approx cpi_{L1} + \sum_{i=2}^{memory} h_i * t_i$$

Knowing that the cpi_0 stays constant gives all the benefits of overlap to the cpi_{stall} term. Thus, the difference between the maximized cpi_{stall} and the $cpi_{overlap}$ will give us an upper bound for the cpi_{stall} term.

$$(8) \quad cpi - cpi_{L1} + cpi_{overlap}^{min} = cpi_{stall}^{upper}$$

The individual components of (8) are shown in figure 7. The upper bound for cpi_{stall} is always below the overall cpi . Also, note that in some cases the approximation of cpi_0 with cpi_{L1} introduced some errors: two of the upper bound values are slightly negative (their first two decimal digits are zero). This is caused by the nature of the code and its problem size.

Figure 7. Upper Bound for cpi_{stall} using the lower bound for $cpi_{overlap}$ and cpi_0 approximation cpi_{L1} . Data are shown for the ASCI benchmarks on Origin 2000 systems.

The two codes in which this phenomenon happens can be considered resident in cache; therefore the amount of time spent stalling is negligible. Thus, in these cases the quantity cpi_{stall} that we want to estimate using our approximation of cpi_0 is of the same order as the error introduced by

the approximation. Looking at the whole picture one can see that this problem happens only in those cases where cpi is smaller than cpi_{L1} . In these cases the measurement of cpi_{stall} is actually not so relevant because it is negligible compared to cpi_0 (whether real or approximated).

The upper bound for cpi_{stall} is shown in figure 8 together with the overall cpi and the maximized cpi_{stall} that we used do compute the lower bound for the overlap. From the chart it is possible to observe how the bounded cpi_{stall} is different from the maximized cpi_{stall} which could be seen as the worst case. Thus, this upper bound gives a tighter estimation of the actual cpi_{stall} ; also, because its value is in all cases less than the overall cpi .

Figure 8. Upper Bound for cpi_{stall} compared with the overall cpi and the maximum cpi_{stall} possible. The upper bound cpi_{stall} is always below the overall cpi . Data are shown for the ASCI benchmarks on Origin 2000 systems.

Figure 9 shows the comparison of the upper bound for cpi_{stall} compared with another possible upper bound derived using the ideal cpi_0 for the Origin 2000 system. This upper bound is computed as follow:

$$(9) \quad cpi_{stall}^{0.25} = cpi - 0.25$$

Figure 9. Upper Bound for cpi_{stall} compared with the overall cpi and a cpi_{stall} computed assuming ideal cpi_0 (the best possible: 0.25). The upper bound cpi_{stall} is always below the cpi_{stall} computed using the ideal cpi_0 . Data are shown for the ASCI benchmarks on Origin 2000 systems.

The data in figure 9 show that the upper bound for cpi_{stall} that is derived using the lower bound for the overlap gives a tighter bound than the one computed using the ideal cpi_0 . The fact that the ideal cpi_0 is independent of any code characteristic, but is dependent only on the architecture component, doesn't enable this bound to capture all the cases well. Using our upper bound we can capture code characteristics providing a tighter upper bound to the actual cpi_{stall} .

Lower bound for cpi_0

A lower bound for cpi_0 can be derived in a way similar to the derivation of the upper bound for cpi_{stall} . The overlap contribution is estimated using our previous result on the lower bound in equation (10).

$$(10) \quad cpi_0^{lower} = cpi - (cpi_{stall}^{max} - cpi_{overlap}^{min})$$

Figure 10. Lower Bound for cpi_0 compared with the overall cpi and cpi_{L1} . The lower bound is always below the overall cpi and cpi_{L1} . Data are shown for the ASCI benchmarks on Origin 2000 systems.

The lower bound for cpi_0 is shown in figure 10 as a function of the overall cpi and cpi_{L1} (where cpi_{L1} is the approximated value measurable for problem size that are primary cache resident). The chart shows that the lower bound is always smaller than the overall cpi and that is always smaller than cpi_{L1} . Also, the lower bound is always greater than the best cpi_0 , which on the system under consideration is 0.25. While 0.25 can be considered as a lower bound, it cannot be considered a tight one since it is independent of any code characteristics, and is only architecture dependent. Instead, the lower bound of equation (10), considers components such as the overall cpi , the stall component, and the overlap, all of which are code and architecture dependent. Thus, in those cases where the stall time is minimal, the lower bound reflects this with a value close to the overall cpi and cpi_{L1} .

7. Conclusion

This paper proposes an empirical technique to compute a lower bound to characterize the effects of overlap. The result produces a tight bound, though not sharp, lower bound. The lower bound obtained using the technique described in section 4 has been applied to characterize the performance of the ASCI benchmarks. Section 5 shows in what proportion the overlap is present

in each code, as well as how it varies with the problem size. A comparison of two different architectures using the lower bound is presented in section 5. The lower bound result has also been applied to compute tight bounds for two quantities that cannot be directly measured. An upper bound for the cpi_{stall} component and a lower bound for the cpi_o component are presented in section 6. The technique to compute a lower bound to quantify the effects of overlap is an empirical one. Based on measured data a lower bound can be computed. Unlike iterative methods, such as those used in [2,9], a lower bound computed as described here is a better approximation of the actual value. While an iterative method such as the least square fit has a margin of error that could be negative or positive, using the lower bound result it is guaranteed that the actual value is going to be greater. The result obtained using this technique could be used in combination with a least square fit method to provide more constraints that could reduce the percentage of error. Future studies will continue to analyze the problem of characterizing the impact of overlap, looking particularly at how different architecture features contribute to the overall overlap. Also, future studies will try to apply this technique to performance prediction.

Acknowledgments

Many thanks to Olaf Lubeck, Yong Luo, and Harvey Wasserman for their precious suggestions and comments. Many thanks also to the Parallel Architecture Team at Los Alamos National Lab, and in particular to Michele Benzi, Jeffrey Hammes, and Adolfo Hoisie for the careful readings and critiques.

References

- [1] Vernon, M.V, Lazowska, E. D., and Zahorjan, J., "An Accurate and Efficient Performance Analysis Technique for Multiprocessor Snooping Cache-consistency Protocols," in Proc. 15th Annu. Symp. Comput. Architecture, Honolulu, HI, June 1988, pp 308-315.
- [2] Lubeck, O.M., Luo, Y., Wasserman, H. J., Bassetti F., "Performance Evaluation of the SGI Origin 2000: A memory centric characterization of LANL ASCI applications", to appear in proc. of Supercomputing'97.
- [3] Luo, Y., Lubeck, O.M., and Wasserman, H. J., "Preliminary Performance Study of the SGI Origin2000," Los Alamos National Laboratory Unclassified Release LA-UR -, 1997.
- [4] Koch, K. R., Baker, R. S. and Alcouffe, R. E., "Solution of the First-Order Form of the 3-D Discrete Ordinates Equation on a Massively Parallel Processor," Trans. of the Amer. Nuc. Soc., 65, 198, 1992.
- [5] W. D. Schulz, "Two-Dimensional Lagrangian Hydrodynamic Difference Equations," Methods in Computational Phys. Vol 3, p1, 1964.
- [6] Zagha, M., Larson, B., Turner, S., and Itzkowitz, M., "Performance Analysis Using the MIPS R10000 Performance Counters," Proc. Supercomputing '96, IEEE Computer Society, 1996.
- [7] Laudon, J. and Lenoski, D., "The SGI Origin: A ccNUMA Highly Scalable Server", Proc. International Symposium of Computer Architecture, ACM, June 1997, Denver, CO.
- [8] Cormen T.H., Leiserson C.E., Rivest R.L. "Introduction to Algorithms" The MIT Press, 1990.
- [9] Lubeck, O.M., Luo, Y., Wasserman, H. J., Bassetti F., "Performance Evaluation of the Origin 2000 on the ASCI Workload: A Memory Centric View", submitted to High Performance Computer Architecture 1998.
- [10] Yeager, K. C., "The MIPS R10000 Superscalar Microprocessor," IEEE Micro, April 1996, pp 28-40.
- [11] "Origin and Onyx Theory of Operation Manual", SGI Tech. Rep. 007-3439-001, Mountain View, CA.

M98001302

Report Number (14) LA-UR--97-3213
CONF-980212--

Publ. Date (11) 199710

Sponsor Code (18) DOE/MA , XF

JC Category (19) UC-905 , DOE/ER

DOE