Simultaneous Partitioning and Frequency Assignment for On-chip Bus Architectures.

Suresh Srinivasan, Lin Li, N. Vijaykrishnan
Department of Computer Science and Engineering
Pennsylvania State University
University Park, PA- 16802
{ssriniva,lili,vijay}@cse.psu.edu

Abstract

In this paper, we provide a methodology to perform both bus partitioning and bus frequency assignment to each of the bus segment simultaneously while optimizing both power consumption and performance of the system. We use a genetic algorithm and design an appropriate cost function which optimizes the solution on the basis of its power consumption and performance. The evaluation of our approach using a set of multiprocessor applications show that an average reduction of the energy consumption by 60% over a single shared bus architecture. Our results also show that it is beneficial to simultaneously assign bus frequencies and performing bus partitioning instead of performing them sequentially.

1. Introduction and Previous work

System on chip architectures have been widely used in recent times as a viable solution to the increasing chip densities, due to the benefits offered by them with respect to improving system performance, cost, power dissipation and reusability. Many shared bus models like the AMBA [1], Coreconnect [2], WishBone [3] have been explored for connecting the various modules in such designs. However, increasing the number of modules connected to a single shared bus tremendously affects the performance and power consumption of such designs. To deal with this problem various designs have been proposed which try to increase the performance and reduce the power consumption of the bus structures [4,5]. Hierarchical bus designs like the Multi-Layer AMBA [6] is one of the ways to counter the problems faced by single shared bus architectures. In such designs the system comprises of multiple buses connected by bridges for inter-bus communications as shown in figure 1. Such a hierarchical designing of bus systems is termed as bus segmentation or bus partitioning. Bus partitioning provides a methodology to keep a check on the growing bus capacitances and increased latencies due to the usage of single shared bus designs. In [7] bus segmentation was introduced for reducing the power consumption of the interconnect architecture. An interesting bus splitting technique for reducing the power consumed by interconnects has been presented in [9]. The results show that bus splitting is preferred over a single shared bus structure due to the benefits it provides with respect to lower power consumption, reliability, smaller driver sizes and larger timing slacks. Most of the bus splitting algorithms are based on clustering the modules present in the system, using the communication profile [8]. The main criteria behind such splitting are to cluster the highly communicating modules together and the low bandwidth modules in other clusters. One of the ways of achieving additional benefits out of the bus segmentation approach would be to operate the various buses on different frequencies. This gives an opportunity to sharply cut down on the power consumption of the low bandwidth module clustered together, by reducing the frequency of the bus they are connected to. This essentially is the motivation behind our work.

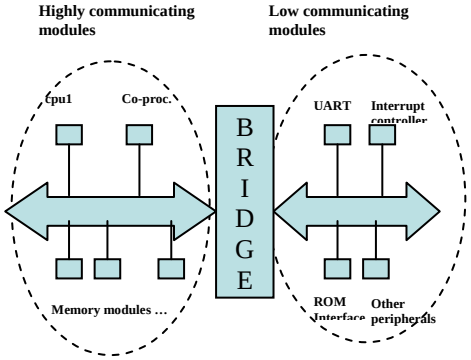

Fig. 1: Hierarchical bus design

In this paper, we provide a methodology for simultaneously performing bus splitting and allocating variable frequency to each of the bus segments, while optimizing on both power consumption and performance of the system. This problem is essentially a clustering problem where given a graph; we are required to partition it into segments, based on multiple optimizing criteria. The nature of the problem ideally suits a genetic algorithm implementation, which solves the problem, provided we design an appropriate cost function. Genetic algorithms have been extensively employed to solve the clustering/partitioning problems in [10]. The genetic algorithm proposed in this paper provides a partitioning methodology and simultaneously

1530-1591/05 $20.00 © 2005 IEEE

assigns appropriate frequencies to each of the bus segments. The solution obtained from our algorithm also provides the topology of the bus segments. The algorithm operates on the criteria of reducing power consumption and increasing the performance of the system.

The rest of the paper is organized as follows. We present the details of the bus partitioning problem in Section 2. The genetic algorithm used to solve the problem is presented in section 3. Section 4 presents the experimental setup followed by the results of our experiments in section 5. The conclusions of the paper are drawn in section 6.

2. Bus Partitioning

Due to the infeasibility of using a single shared bus for many resources, bus segmentation has evolved as a natural alternative solution to the problem. The bus segmentation problem comprises of solving a graph partitioning problem which, given the communication information splits the architecture into smaller bus segments satisfying the constraints imposed. The constraints typically are the performance improvements and the power consumed by any architecture. Such splitting of the bus could be performed in many ways differing in the number of clusters and the way the segments are connected (topology) as shown in figure 5. However, increasing the segmentation of bus is restricted due to the cost of introducing bridges connecting the buses both with respect to power consumption and performance. Both the power consumption and the performance of a bus segment are dependant on the operating frequency of the bus. The power consumption of a bus reduces with the reduction in bus frequency; however this also reduces the performance of the system. This gives an opportunity to operate the buses at different frequencies to optimize on the power consumption and the performance of the system. Bridge in AMBA is unidirectional [6], with the master and the slave having different clocks as shown in figure 2. Our approach of bus partitioning would be applicable on such architectures.

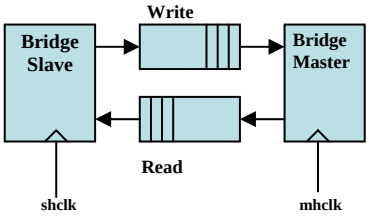

Fig 2: Bridge design in AMBA Multilayer implementation.

3. The Genetic Algorithm

The input to the genetic algorithm is a graph capturing the communication profile for a given architecture as shown in figure 3. Such a communication graph provides us with the information about the transactions in any given application along with the timing of the transactions. The nodes of the graph represent the occurrence of any task and the edges depict the ordering information of the tasks. These tasks are essentially the operation carried out on various processors for the mentioned number of time cycles. The numbers associated with the edges depict the communication among the tasks.

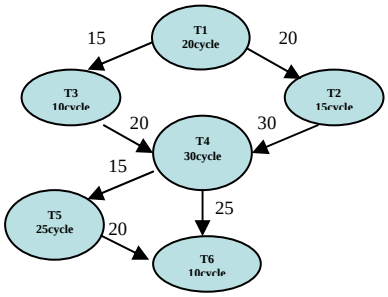

Fig.3: Communication Graph

The goal of the genetic algorithm is to determine an optimal topology for connecting the components and at the same time assign appropriate frequencies to each of the buses. The inputs to the genetic algorithm are (a) the various topologies that we are going to enumerate on (b) the communication profile graph, which provides the information about communication among the various components to be connected to the bus (c) the range of frequencies that we want to assign to each of the buses. We need to have a set of frequencies, which we would be enumerating while analyzing the solution.

3.1. Working of the GA

Chromosome description*:* The chromosome is designed in a way that it captures the clustering information, the frequencies associated with the clusters and the topology for the given number of clusters. The chromosome in our proposed scheme is an array of integers A[0…N+K+1], where N is the number of nodes in the communication graph and K is the maximum number of clusters that we might have such that:

A[i] = t, for 0<=i < N, indicates that the i^{th} node is assigned to the cluster t.

A[i] = x, for N<=i<N+K, means that the bus for the $(i-N)^{th}$ cluster operates in frequency x.

The last value is a number associated with the type of topology. We assign a number to each of the topologies that are possible for a given number of clusters. The inclusion of this value in our chromosome helps us to enumerate the various topologies possible with a given number of clusters.

Crossover operator*:* The crossover function requires us to select two parents for mating. In our algorithm we use a random selection scheme. This enables the

exploration of the solution space in an efficient manner. The crossover operation performs the mating of two parents giving birth to two new chromosomes. The way we perform such an operation is shown in the figure 4. As shown in the figure, we perform the crossover of the two parents separately on the first N and the next K elements. The last element, which defines the topology of the architecture encoded in the solution, is determined randomly for each of the offspring produced by crossover. Note that all the chromosomes always have K frequencies encoded in them regardless of the number of clusters proposed by that chromosome. However, only the relevant frequencies are used in the estimation of the fitness values. This ensures that the crossover operator always generates chromosomes in consistent format. However, there are cases where unidirectional nature of the bridges does not support the segmentation proposed by our solution. In such cases, the fitness function rates the solution as invalid by assigning its fitness value to 0.

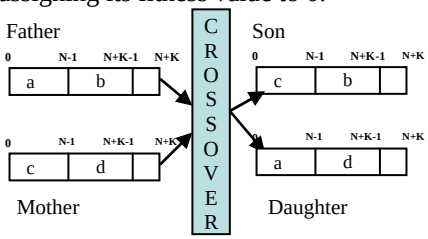

Fig. 4: Crossover operation

Mutation operator*:* The primary motive behind the mutation operator should be to change the entire topology of a solution including the number of clusters. The way this is achieved is by randomly increment/decrementing the values from [1-N], which would surely increase or decrease the total number of clusters in the system and then randomly assigning the frequencies for each of the clusters. Finally, we should also randomly assign the topology for the solution.

Fitness function*:* The fitness function captures the two optimizing criteria, which are the power and performance of a proposed solution.

Performance of a chosen architecture is given by the completion time of any given application on the architecture. Note that, we cannot base the performance numbers on the number of cycles as the buses may be operating on different frequencies and therefore the number of cycles is a misleading number in such a case. The completion time of a given application on a given architecture is primarily dependant on the inter-cluster communication and intra-cluster communication. The clustering algorithm should try to maximize the intra-cluster communication and minimize the inter-cluster communication. However at the same time the completion time should also be minimized.

The calculation of intra-cluster communication cost is relatively easy, given the communication profile. This was done in a similar manner as presented in [8]. However, analysis of cost for inter-cluster communication required us to account for the communication overhead due to the presence of the bridges and the fact that the communication might happen across two different clock domains. Whenever a master requests a communication burst across the bridge, there is a minimum communication overhead of 2 cycles for synchronizing. Note that the two cycles are with respect to the slower clock that the bus is attached to. The completion time for such a communication is also calculated based upon the frequency of the slowest bus among the buses through which the communication occurs. The fitness value for performance is calculated as:

$$F_{per} = 1 - \frac{CT}{CT_{base}}$$

Where F_{per} is the fitness value of the solution with respect to the performance of the system. CT is the completion time of the application on the architecture proposed by the current solution and CT_{base} is the completion time of the application on a shared bus operating at minimum frequency. The higher the completion time the lower the solution is rated with respect to performance.

The power consumption of the given topology could be obtained as a split of the power consumption of the buses and the power consumed by the bridges. The following equation shows the way bridge power is calculated given G_{swun} and G_{swu} are the number of gates in the bridge that switch when the bridge is not used and used respectively and *used* is the fraction of time when the bridge is used.

$$BridgePow \propto G_{swun} * (1 - used) + G_{swu} * used$$

The power consumed by the buses was calculated in similar manner as in [9] paper where the interconnect power is directly proportional to the frequency at which the bus is operated on and the capacitance of the bus. Since the genetic algorithm operates on the principle of comparing the solution we just need an abstract model that captures all the factors that the power is depending upon, and not the exact power numbers. The average power consumption is therefore calculated as:

$$Power = bridgepower + k * C_{bus} * f$$

The capacitance C_{bus} of the bus is calculated using the equation provided in [9], f is the operating frequency of the bus and k is a constant taking care of other factors that power is dependant on. The fitness value for power is computed as shown in the following equation. The maximum power consumption *MaxPower* is the sum of power consumed by a single shared bus operating at

maximum frequency and the maximum power that could be consumed by bridges.

$$F_{pow} = 1 - \frac{Power}{MaxPower}$$

The following equation shows the manner in which the final fitness value is calculated.

$$Fitness = (\alpha * F_{per} + \beta * F_{pow})/(\alpha + \beta)$$

As shown in the equation the final fitness value is calculated as a weighted average of the performance fitness value and the power fitness values. The value of the weights α and β are determined based on the importance of power and performance in the system. We also tested the quality of our solutions by using the fitness function as the product of F_{per} and F_{pow}.

4. Experimental Setup

The genetic algorithm was coded in C language using the Genetic Algorithm Utility Library [11]. We generated the communication graph for a multi-processor benchmark circuit comprising of multiple processor cores communicating with various memory banks. The communication information was obtained by using the tool MP_Simplesim [14] which is a multi-processor implementation of the simplescalar tool [12]. We used a set of benchmarks from SPLASH-2 suite [15]. The communication profile for the benchmark designs were obtained using a configuration of 4 processors and 8 memory banks as well as with an architecture having 8 processors and 8 memory banks. The communication profile provided the detailed timing information of when a processor accesses any memory module. Based on the communication profile the communication graph of the transactions was created.

The various topologies that the algorithm tries to generate are shown in figure 5. We have chosen a maximum of three clusters because of the limited number of modules in our benchmarks. Increasing the number of clusters will increase the power consumption of the architecture with limited performance benefits. The circles in the figure depict a bus segment which has various components connected to it. The edges between the circles depict the presence of a bus. Since bridges are unidirectional they are shown by directed edges connecting the circles where, the direction of the edge determines the master and the slave of the bridge.

5. Results

The genetic algorithm was implemented on an Intel Xeon dual processor machine. The size of the population was chosen as 200 and the algorithm was allowed to run for 5000 generations. We chose to operate on a frequency range of 50 to 250 MHz and enumerated on frequency values that are multiples of 25. These are the frequencies that the algorithm tries to assign to the different buses. The communication traces were obtained for the benchmark designs for 2 million cycles using a single shared bus.

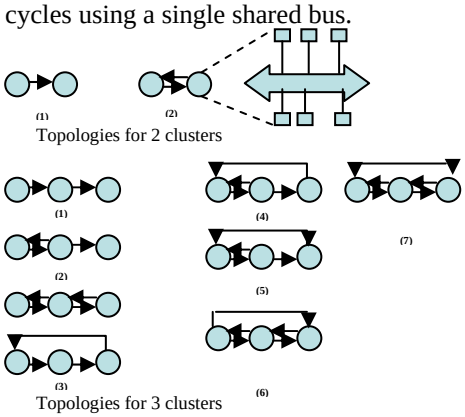

Fig 5: The topologies used by our algorithm

We select the *radix* benchmark as a representative to perform extensive analysis of the solutions generated by our algorithm with varying of fitness functions. We investigate the various interconnect topologies chosen by our algorithm, by varying the weights associated with the power fitness value ('a') and the performance fitness value ('b'), while calculating the final fitness function. Note that the weights are the metrics to determine the criticality of power or performance in the system. The various fitness functions and the results obtained are shown in table 1. The results show that the optimal topology for the bus design is chosen to be segmented topology over a single shared bus topology in all the cases. Another observation that could be made from the table is the fact that as the weights associated with the power fitness values are increased the solution tends to have its buses operating at lower frequencies. We also show the completion times of our final solution. The completion time on a single bus operating at highest frequency of 250MHz is 0.08 sec. The solution provided by our algorithm when performance is weighted high performs better with respect to the single shared bus design operating at the highest frequency. Note that even when we rate power more than performance the final solution performs better than the single shared bus design in net energy consumption. We can observe an average reduction in energy consumption by 49% as compared to the single bus operating at 250MHz.

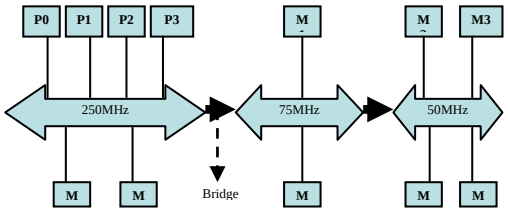

Fig 6: Bus partitioning and frequency assignment for radix benchmark provided by our approach for fitness function a*b.

Function	seg	top	freq	Reduction in Energy over single bus at 250MHz(%)	Completion time
a	2	1	50,50	24.71	0.300999
b	2	2	250,250	6.27	0.070753
(a + b)/2	2	1	250,50	60.06	0.088734
(a+2*b)/3	3	2	250,75,50	62.13	0.078823
(a+3*b)/4	3	1	250,125,50	61.82	0.072821
(2*a+b)/3	3	1	175,75,50	54.98	0.125046
(3*a+b)/4	3	1	125,75,50	62.16	0.139672
a*b	3	1	250,75,50	64.84	0.083332

Table 1: The fitness function used where 'a' and 'b' are the power and performance fitness values respectively for the *radix* benchmark design.

Figure 6 presents the solution determined by our algorithm when we choose the fitness function as the product of the power and performance fitness values. The solution has three bus segments where only one of the segments is operated at the highest frequency while the other two buses are operated at lower frequencies of 75MHz and 50MHz respectively. The reason for our solution to arrive at this solution can be explained by analyzing the communication traces of the benchmark circuit. The *radix* benchmark had all the processors accessing the memory banks *M5, M6* significantly more than the rest of the memory modules. This provides an opportunity to the genetic algorithm for placing the memory modules other *M5* and *M6* in different bus segments that could operate on lower frequencies and thereby help in saving power without significant impact on performance. The highly accessed memory modules were placed in the first cluster along with the processors and the least used ones are placed in the last cluster. This reflects the quality of our approach to in being able to reduce power consumption on buses supporting low bandwidth devices by adjusting the frequencies. We tested the solution presented by our algorithm by fixing the frequencies at which the buses operate and compared the results obtained when the algorithm performed variable frequency assignment. The results as shown in figure 7 indicate that variable frequency assignment results in considerable energy savings than the case when the algorithm is forced to assign a fixed frequency.

Table 2 presents a comparison of the energy and performance of the solutions generated when we perform simultaneous frequency assignment with the solutions generated by first performing the clustering at a fixed frequency followed by frequency allocation. Our results show that solutions generated by performing simultaneous frequency assignment are better than that of sequential bus clustering and bus assignment approach.

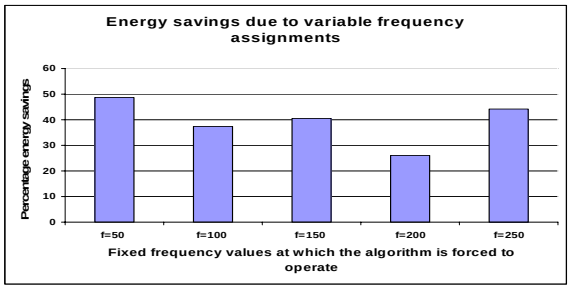

Fig 7: Energy savings of the solutions generated when the algorithm is forced to assign fixed frequency than when it performs simultaneous variable frequency assignment.

Frequency used during clustering phase for sequential approach	Energy reduction due to simultaneous approach over seq. approach.	Completion time for sequential approach	Completion time for simultaneous approach
f=50	15.61	0.086882	0.083332
f=100	13.03	0.084222	0.083332
f=150	14.182	0.088732	0.083332
f=200	16.1252	0.085223	0.083332
f=250	17.532	0.083770	0.083332

Table 2: Comparing algorithms with simultaneous and non-simultaneous frequency assignment.

Benchmark	Seg.	Top.	Frequencies
barnes(4,8)	2	2	175,175
cholesky(4,8)	2	2	150,150
Radix(4,8)	3	1	200,75,50
raytrace(4,8)	2	1	200,50
water(4,8)	2	2	125,125
barnes(8,8)	2	2	100,100
cholesky(8,8)	2	2	150,175
Radix(8,8)	3	2	200,150,50
raytrace(8,8)	2	1	200,50
Water(8,8)	2	2	150,150

Table 3: Solutions obtained for various benchmark designs.

We used the fitness function *a*b* to determine the optimal topologies for the rest of the benchmark designs. Table 3 shows the results of our implementations on various benchmark circuits. The values (m, n) which follow each of the benchmark designs in the table depict the number of processors and the number of memory banks in the system, respectively. Most of the benchmarks result in bus architectures with two-partitions and fully connected topology due to the similar communication profiles of these benchmark designs. Figure 7 shows a plot of the energy savings obtained by using the final solution over the single shared bus architecture operating at the frequency of 50MHz and 250MHz for various benchmark designs. Note that the final solution is able to achieve energy savings of nearly 60% for the benchmark designs.

Figure 8 shows the completion times of our final solution compared with completion time of a single shared bus design operating at frequency 250MHz. The

solution generated by our algorithm outperforms the single shared bus solution for almost all the benchmark. In case of the *radix* benchmark, the final completion time is larger than the completion time of single shared bus operating at 250MHz. This is due to the fact that the final solution is selected based on both the power consumption and the performance metrics. In case of the *radix* benchmark, the significant power savings in the final solution compensates for the slight performance penalty. The percentage reduction in the energy consumption of the final solution is infact the largest in this benchmark. Since the solution provided by our algorithm is significantly better with respect to performance of the single shared by operating at the least frequency of 50MHz, we omit those results.

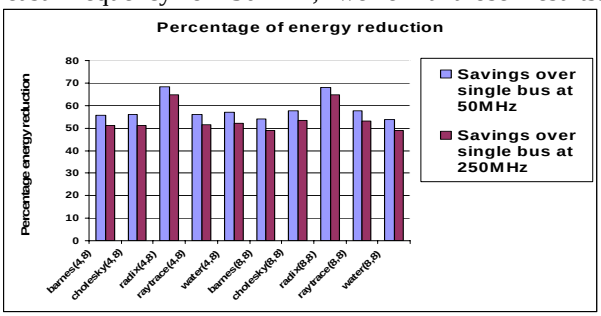

Fig. 8: Percentage reduction in energy consumption of the final solution over the single shared bus solution.

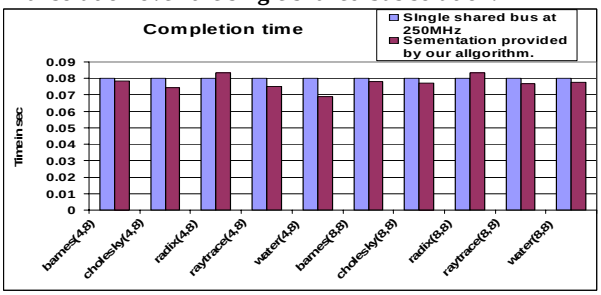

Fig. 9: The completion time of our proposed solution as compared to the single shared bus architecture operating at 50MHz and 250MHz.

6. Conclusion

We provide a genetic algorithm based methodology to perform both bus partitioning and bus frequency assignment to each of the bus segment simultaneously while optimizing both power consumption and performance of the system. The generation of the solution is directed by the power and performance constraints imposed by us. The results present some interesting tradeoffs in the power and performance numbers. The prime feature of the algorithm is its adjustable nature to assigning importance to power and performance based on the requirements of the system. The evaluation of our approach using a set of multiprocessor applications show that an average reduction of the energy consumption by 60% over a single shared bus architecture. Our results also show that it is beneficial to simultaneously assign bus frequencies and performing bus partitioning instead of performing them sequentially.

Acknowledgements

This work was supported in part by NSF CAREER AWARD 0093085 and grants from SRC and GSRC/PAS.

References

[1] Flynn. AMBA: enabling reusable on-chip designs, *IEEE Micro,* 1997.

[2] CoreConnect Bus Architecture. http://www.chips.ibm.com /products/coreconnect.

[3] WishBone Specification. http://www.silicore.net/ wishbone.htm.

[4] K. Lahiri, G. Lakshminarayana, and A. Raghunathan. LOTTERYBUS: A new communication architecture for high-performance system-on-chip design. *In proceedings of Design Automation Conference,* June 2001.

[5] B.Cordan. An efficient bus architecture for system-on-chip design. *In proceedings of Custom Integrated Circuits Conference,* 1999.

[6] AMBA Specification (rev2.0) and Multi layer AHB specification, Arm: http://www.arm.com, 2001.

[7] W.B. Jone, J.S. Wang, H. Lu, I.P.HSU and J.Y.Chen. Segmented Bus design for low-power systems. *IEEE transactions on VLSI Systems,* March 1999.

[8] K. Lahiri, A. Raghunathan and S. Dey. Design space exploration for optimizing on-chip communication architectures. *IEEE transactions on Computer-Aided Design of Integrated Circuits and Systems,* June 2004.

[9] C-Ta Hsieh and M. Pedram. Architectural power optimization by bus splitting, *Design, Automation and Test in Europe,* 2000.

[10] H. Maini, K. Mehrotra, C. Mohan and R. Sanjay. Genetic algorithms for graph partitioning and incremental graph partitioning, *In proceedings of 5th International conference on Supercomputing,* 1994.

[11] GAUL: Genetic Algorithm Utility Library. http://gaul.sourceforge.net.

[12] D. Burger, and T.M. Austin. The simplescalar toolset version 2.0. Technical report 1342, Department of Computer Sciences, The University of Texas at Austin, June 1997 .

[13] D. Liu and C. Svenoson. Power consumption estimation in CMOS VLSI chips. *IEEE Journal on Solid State Circuits,* June 1994.

[14] N. Manjikian. Multiprocessor enhancement of simplescalar toolset. *ACM SIGARCH Computer Architecture News,* 2001.

[15] S. C. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The splash-2 programs: characterization and methodological considerations. *In ISCA-22, pp. 24,* 1995.