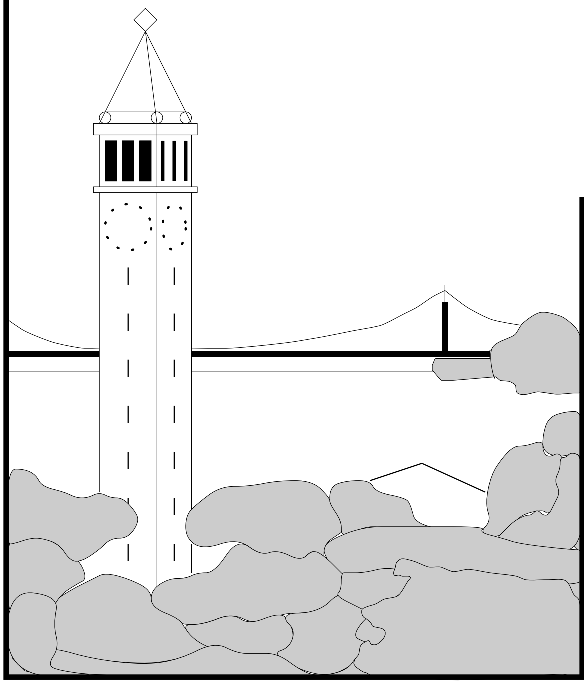
A model for speedup of parallel programs

Allen B. Downey



Report No. UCB/CSD-97-933

January 1997

Computer Science Division (EECS)
University of California
Berkeley, California 94720

A model for speedup of parallel programs

Allen B. Downey *

January 1997

Abstract

We propose a new model for parallel speedup that is based on two parameters, the average parallelism of a program and its variance in parallelism. We present a way to use the model to estimate these program characteristics using only observed speedup curves (as opposed to the more detailed program knowledge otherwise required). We apply this method to speedup curves from real programs on a variety of architectures and show that the model fits the observed data well. We propose several applications for the model, including the selection of cluster sizes for parallel jobs.

1 Introduction

Speedup models describe the relationship between cluster size and execution time for a parallel program. These models are useful for:

Modeling parallel workloads : Many simulation studies use a speedup model to generate a stochastic workload. Since our model captures the behavior of many real programs, it lends itself to a realistic workload model.

Summarizing program behavior : If a program has run before (maybe on a range of cluster sizes), we can record past execution times and use a speedup model to summarize the historical data and estimate future execution times. These estimates are useful for scheduling and allocation.

Inference of program characteristics : The parameters of our model correspond to measureable program characteristics. Thus we hypothesize that we can infer these characteristics by fitting our model to an observed speedup curve and finding the parameters that yield the best fit.

Our speedup model is a non-linear function of two parameters: A, which is the average parallelism of a job, and σ, which approximates the coefficient of variation of parallelism. The family of curves described by this model spans the theoretical space of speedup curves. In [7], Eager, Zahorjan and Lazowska derive upper and lower bounds for the speedup of a program on various cluster sizes (subject to simplifying assumptions about the program's behavior). When $\sigma = 0$, our model matches the upper bound; as σ approaches infinity, our model approaches the lower bound asymptotically.

This model might be used differently for different applications. In [5] and [6] we use it to generate the stochastic workload we use to evaluate allocation strategies for malleable[1] jobs. For that application, we choose the parameters A and σ from distributions and use them to generate speedup curves. In this paper, we work the other way around — we use observed speedup curves to estimate the parameters of real programs. Our goal here is to show that this model captures the behavior of real programs running on diverse parallel architectures. This technique is also useful for summarizing the speedup curve of a job and interpolating between speedup measurements.

1.1 Related work

In [4], Dowdy proposed a speedup model based on a program with a sequential component of length c_1 and a perfectly parallel component of length c_2. The execution time, $T(n)$, of such a program is $T(n) = c_1 + c_2/n$, where n is the number of processors.

Chiang *et al.* [3] derive from this a model of speedup with the form $S(n) = (1+\beta)n/(n+\beta)$, where the parameter β is a program characteristic that varies from 0 for a sequential program to infinity for a program with linear speedup. Several subsequent studies have been based on

*EECS — Computer Science Division, University of California, Berkeley, CA 94720 and San Diego Supercomputer Center, P.O. Box 85608, San Diego, CA 92186. Supported by NSF grant ASC-89-02825 and by Advanced Research Projects Agency/ITO, Distributed Object Computation Testbed, ARPA order No. D570, Issued by ESC/ENS under contract #F19628-96-C-0020. email: downey@sdsc.edu, http://www.sdsc.edu/~ downey

[1] A malleable job is a parallel program that can run on a range of cluster sizes. The allocation strategy is the part of the scheduler that chooses the cluster size for each malleable job.

this model [10] [14]. Brecht and Guha use a variation of this model that imposes an upper bound on the speedup of some jobs [1] [9].

One problem with this model is that the parameter β has little semantic content. Thus, it is not clear how to use observations of a real program to find the value of β or how to choose a distribution of values that describes a real workload. As a result, workload models based on Dowdy's speedup model have tended to overestimate the parallelism available in codes executing in supercomputing environments. With our model, we have been able to use observations of the workload at the San Diego Supercomputer Center to infer the parameters of real workloads [5] [6].

Sevcik [16] and Ghosal *et al.* [8] have proposed alternative models based on more detailed program information. These models have many free parameters, and therefore provide no way to infer program characteristics from observed behavior. Furthermore, it would be difficult to specify the range of these parameters in a real workload.

Smirni *et al.* [17] use a speedup model with the following functional form: $S(n) = (1-\tau^n)/(1-\tau)$ with $0 \leq \tau \leq 1$. The motivation for this model is to facilitate analysis. Again, the parameter τ has no semantic content.

No prior study has demonstrated that a proposed model describes the behavior of real programs. Chakrabarti *et al.* [2] propose a model for efficiency of data parallel tasks; they use measurements of ScaLAPACK programs to validate this model.

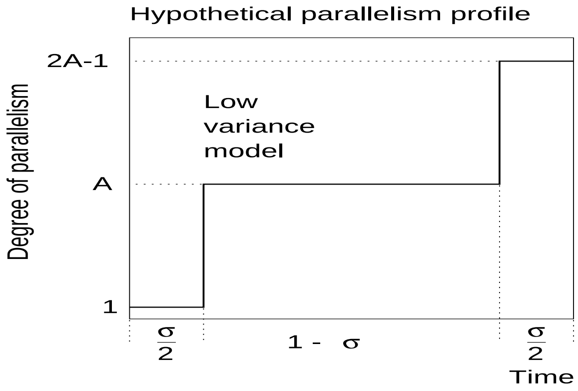
Many of the allocation strategies that have been proposed for malleable jobs assume that the scheduler knows the average parallelism of all jobs [16] [8] [15] [17] [3] [12] [1]. Thus all of these strategies require that the parallelism profile of the program be known, or that A (and maybe V) can be calculated by other means. Our model may provide a way to derive these characteristics.

2 The model

The design goal for our speedup model is to find a family of speedup curves that are parameterized by the average parallelism of the program, A, and the variance in parallelism, V. To do this, we construct a hypothetical *parallelism profile*[2] with the desired values of A and V, and then use this profile to derive speedups. We use two families of profiles, one for programs with low variance, the other for programs with high variance.

[2] The parallelism profile is the distribution of potential parallelism of a program[16].

a)



b)

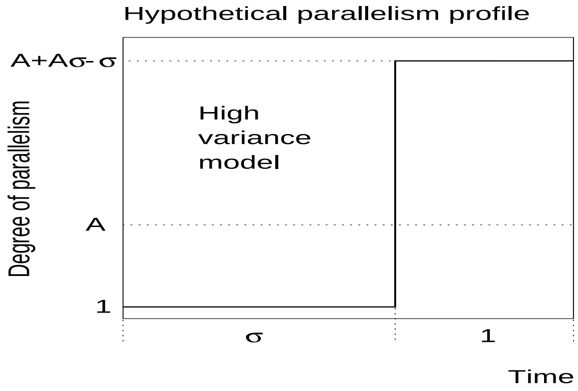


Figure 1: The parallelism profile for (a) the low variance speedup model and (b) the high variance speedup model.

2.1 Low variance model ($\sigma \leq 1$)

Figure 1a shows a hypothetical parallelism profile for a program with low variance in degree of parallelism. The parallelism is equal to A, the average parallelism, for all but some fraction σ of the duration ($0 \leq \sigma \leq 1$). The remaining time is divided between a sequential component and a high-parallelism component (with parallelism chosen such that the average parallelism is A). The variance of parallelism is $V = \sigma(A-1)^2$.

A program with this profile would have the following run time as a function of cluster size:

$$T(n) = \begin{cases} \frac{A-\sigma/2}{n} + \sigma/2 & 1 \leq n \leq A \\ \frac{\sigma(A-1/2)}{n} + 1 - \sigma/2 & A \leq n \leq 2A-1 \\ 1 & n \geq 2A-1 \end{cases} \quad (1)$$

where n is the cluster size (number of processors). Thus $T(1) = A$ and $T(\infty) = 1$. The speedup, $S(n) = T(1)/T(n)$, is

$$S(n) = \begin{cases} \frac{An}{A+\sigma/2(n-1)} & 1 \le n \le A \\ \frac{An}{\sigma(A-1/2)+n(1-\sigma/2)} & A \le n \le 2A-1 \\ A & n \ge 2A-1 \end{cases} \quad (2)$$

2.2 High variance model ($\sigma \ge 1$)

In the previous section, the parameter σ is bounded between 0 and 1, and thus the variance of the parallelism profile is limited to $V = (A-1)^2$ when $\sigma = 1$. In this section, we propose an extended model in which sigma can exceed 1 and the variance is unbounded. The two models can be combined naturally because (1) when the parameter $\sigma = 1$, the two models are identical, and (2) for both models the variance of the parallelism profile is $\sigma(A-1)^2$.

From this latter property we derive the semantic content of the parameter σ — it is approximately the square of the coefficient of variation of parallelism, CV^2. This approximation follows from the definition of coefficient of variation, $CV = \sqrt{V}/A$. Thus, CV^2 is $\sigma(A-1)^2/A^2$, which for large A is approximately σ.

Figure 1b shows a hypothetical parallelism profile for a program with high variance in parallelism. The profile consists of a sequential component of duration σ and a parallel component of duration 1 and potential parallelism $A + A\sigma - \sigma$. A program with this profile would have the following run time as a function of cluster size:

$$T(n) = \begin{cases} \sigma + \frac{A+A\sigma-\sigma}{n} & 1 \le n \le A + A\sigma - \sigma \\ \sigma + 1 & n \ge A + A\sigma - \sigma \end{cases} \quad (3)$$

Thus $T(1) = A(\sigma+1)$ and $T(\infty) = \sigma + 1$. The speedup is

$$S(n) = \begin{cases} \frac{nA(\sigma+1)}{\sigma(n+A-1)+A} & 1 \le n \le A + A\sigma - \sigma \\ A & n \ge A + A\sigma - \sigma \end{cases} \quad (4)$$

Figure 2 shows a set of speedup curves for a range of values of σ. When $\sigma = 0$ the curve matches the theoretical upper bound for speedup — bound at first by the "hardware limit" (the 45 degree line) and then by the "software limit" (the average parallelism A). As σ approaches infinity, the curve approaches the theoretical lower bound on speedup [7]:

$$s_{low}(n) = \frac{An}{A+n-1} \quad (5)$$

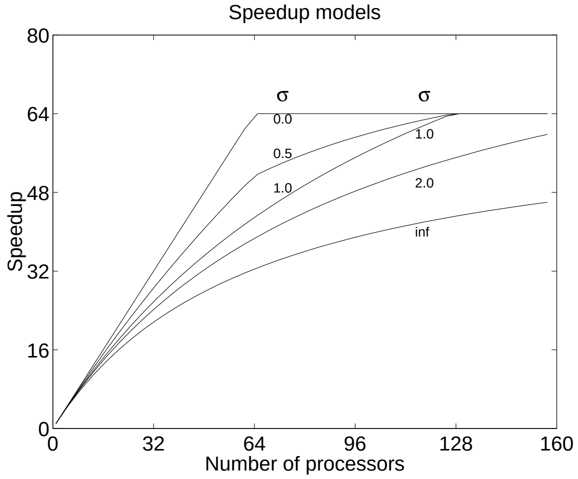


Figure 2: Speedup curves for a range of values of σ.

2.3 Calculating the knee

Several authors have proposed the idea that an optimal allocation for a program is the one the maximizes the power, Φ, which is defined as the product of the speedup and the efficiency, $e(n) = s(n)/n$. Thus, $\Phi = s^2/n$. We search for the value of n that maximizes Φ by finding local maxima where $\frac{d\Phi}{dn} = 0$:

$$\frac{d\Phi}{dn} = \frac{2ns\frac{ds}{dn} - s^2}{n^2} = 0$$

$$2ns\frac{ds}{dn} = s^2 \quad (6)$$

The speedup curves proposed in Equations 2 and 4 have the functional form $s(n) = \kappa n / u(n)$, where κ is a constant with respect to n, and $u(n)$ is some function of n. Substituting this functional form into Equation 6 yields:

$$2n \underbrace{\frac{\kappa n}{u}}_{s} \underbrace{\kappa \frac{u - n\frac{du}{dn}}{u^2}}_{\frac{ds}{dn}} = \underbrace{\frac{\kappa^2 n^2}{u^2}}_{s^2}$$

$$u = 2n\frac{du}{dn} \quad (7)$$

Then, using Equation 2, we can find the "knee" of the low-variance speedup curve:

$$\begin{aligned} u &= \sigma(A - 1/2) + n(1 - \sigma/2) \\ \frac{du}{dn} &= 1 - \sigma/2 \\ n^* &= \frac{\sigma(A-1/2)}{1-\sigma/2} \end{aligned} \quad (8)$$

where n^* is the optimal cluster size. Using Equation 4 for the high variance model:

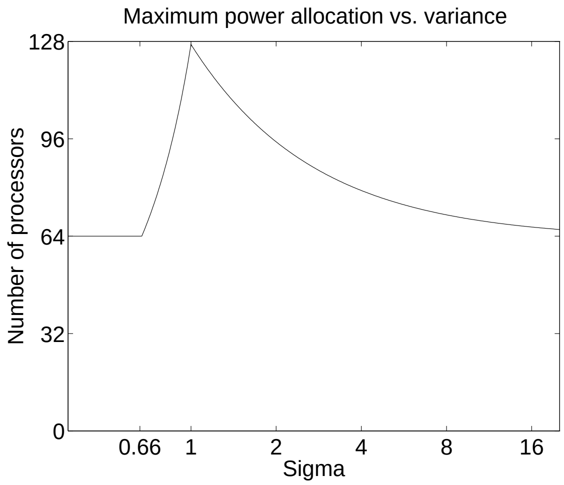


Figure 3: The optimal allocation for a range of values of σ. The average parallelism, A, is 64.

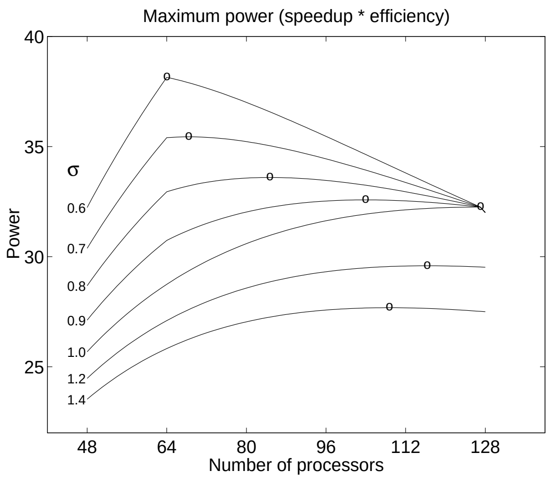


Figure 4: Power as a function of cluster size for several values of σ. The small circle on each curve indicates the point of maximum power. The average parallelism, A, is 64.

$$\begin{aligned} u &= \sigma n + A(\sigma+1) - \sigma \\ \frac{du}{dn} &= \sigma \\ n^* &= \frac{A(\sigma+1)-\sigma}{\sigma} \end{aligned} \tag{9}$$

Figure 3 shows the optimal cluster size, n^*, as a function of σ, with A fixed at 64. When σ is less than $2A/(3A-1)$, which is approximately 2/3, the point of maximum power is $n^* = A$, which is in accord with the heuristic that the number of processors allocated to a job should be equal to its average parallelism [7][8]. It also agrees with the colloquial interpretation of the "knee" of the curve — a discontinuity in its slope.

But as the value of σ approaches 1, n^* increases quickly to $2A - 1$. For larger values of σ, it decreases gradually and approaches $A - 1$ asymptotically. This result is both surprising and discouraging: surprising because it violates the intuition that the optimal allocation for a job should decline monotonically as the variance in parallelism increases [16], and discouraging because the rapid change in the point of maximum power suggests (1) that the "knee" of the curve is not well-defined — small changes in observed speedups might lead to drastically different allocations, and (2) that the assumption that the point of maximum power is an optimal allocation is probably wrong. In [5] we confirm that scheduling strategies that attempt to allocate this "optimal" cluster size do not perform as well as other strategies, including one that simply allocates A processors to each job.

Figure 4 confirms that the local maxima from Equations 8 and 9 are in fact the global maxima over all feasible cluster sizes.

3 Estimating parameters

Given a set of observed speedups s_i for cluster sizes n_i, we would like to estimate values of the parameters A and σ that minimize the sum of squared differences between the observed values and the fitted values (calculated by Equations 2 and 4). In other words, we would like to minimize

$$\chi^2(A,\sigma) = \sum_i [s_i - s(n_i; A, \sigma)]^2 \tag{10}$$

where $s(n_i; A, \sigma)$ is the modeled speedup of a program with average parallelism A and variance σ, running on n_i processors.

The Levenberg-Marquardt method [11] performs this minimization by a variation of the conjugate gradient method that takes advantage of the form of the object function, χ^2.

Starting with an initial estimate for the parameters A and σ, we iteratively calculate improved estimates based on the value of χ^2 and its derivatives with respect to A and σ. These derivatives are

$$\frac{\delta\chi^2}{\delta A} = -2\sum_i [s_i - s(n_i; A, \sigma)] \frac{\delta s(n_i; A, \sigma)}{\delta A} \tag{11}$$

$$\frac{\delta\chi^2}{\delta \sigma} = -2\sum_i [s_i - s(n_i; A, \sigma)] \frac{\delta s(n_i; A, \sigma)}{\delta \sigma} \tag{12}$$

where the partial derivatives $\frac{\delta s}{\delta A}$ and $\frac{\delta s}{\delta \sigma}$ are derived from Equations 2 and 4.

The Levenberg-Marquardt method chooses adaptively between the conjugate gradient method (which converges quickly in the vicinity of the minimum) and the method of steepest descent (which is robust when the object function is ill-behaved). In practice, we have found that this method is robust and converges quickly for a variety of speedup curves. We have used this method to estimate parameters for many speedup curves reported from real programs on a variety of architectures. The next section discusses these results.

4 Fitting observed speedups

We developed our speedup model with two applications in mind:

Workload modeling : We would like to know whether our model captures the behavior of parallel scientific applications, and if so, what values of the parameters A and σ are typical of the workloads in supercomputing environments.

Summarizing program behavior : For purposes of scheduling parallel jobs, we can use a database of past execution times to predict future execution times. If our model fits observed speedup curves well, then we can greatly compress this database by recording, for each program and problem size, only the two parameters A and σ.

In order to evaluate the usefulness of our model, we have collected reported speedup curves from numerous scientific applications running on a variety of parallel architectures. In general, we have found that our model is capable of summarizing the behavior of most real programs. In the next two sections, we present some of this data, point out cases in which our model fails, and suggest ways of dealing with these failures.

4.1 NAS benchmarks

The Numerical Aerospace Simulation Facility (NAS) at NASA Ames Research Center has compiled a suite of benchmarks intended to be representative of computational fluid-dynamic codes. The original NAS Parallel Benchmarks (NPB 1) were algorithmic specifications of eight computations. NPB 2 consists of implementations of four of those computations in Fortran 77 with MPI. NAS has reported timings of these codes on four different distributed-memory computers with three different problem sizes. The programs are:

LU : Solves Navier-Stokes equations in 3-D using LU decomposition and successive over-relaxation (SSOR). Due to the internal structure of the code, it requires power-of-two cluster sizes.

SP and BT : Both solve Navier-Stokes equations in 3-D, based on a Beam-Warming factorization. In SP, the resulting system is scalar pentadiagonal; in BT, it is block triangular. In both cases, the system is solved by Gaussian elimination without pivoting. The decomposition used (3-D multipartitioning) requires cluster sizes that are perfect squares.

MG : Solves Poisson's equation using a V-cycle multigrid algorithm. This code requires power-of-two cluster sizes.

The codes were run on an IBM SP2, an SGI Power Challenge Array, a Cray Research T3D and an Intel Paragon. The three problem sizes are named Class A (the smallest), Class B and Class C (the largest). For more details about the benchmarks and test architectures, see [13] and http://www.nas.nasa.gov/NAS/NPB.

Figure 5 shows the speedups observed on the SP2 and our estimated parameters and speedup curves. In all cases, the fitted curve matches the observed data well. The only exception is the surprisingly bad performance of the SP class C benchmark on 36 processors. This datum may be in error, or it may be a consequence of a memory-system phenomenon like a cache collision.

Memory requirements prevent some Class C benchmarks from running on small cluster sizes. In these cases we normalize the observed speedup values before estimating parameters — in effect, we treat the smallest feasible cluster size as a single processing unit. Thus, the efficiency on the smallest cluster size is defined to be 1. We have observed that this normalization does not degrade the goodness of fit of the model.

For each benchmark, we expect the estimated average parallelism, A, to increase with problem size. In fact, this is true of LU, MG and SP, but not true of BT. The estimated value of A for class C is smaller than that for class B. This example illustrates one failure mode of our model: if the observed timings exhibit linear or near-linear speedup, then our model has no way of inferring the average parallelism of the code. Any value of A (greater than the largest observed speedup) would yield the same goodness of fit. It happens that the curve-fitting technique we use converges on $A = s_{max}$, where s_{max} is the largest observed speedup.

Figure 6 shows the speedups observed on the T3D. Because the amount of memory per node is smaller on this machine than on the SP2, none of the benchmarks were able to run on a single processor. Thus all of our curve fits are based on normalized data.

Unlike the SP2, which can allocate arbitrary cluster sizes, the T3D can only allocate power-of-two cluster sizes. As a result, the performance of BT and SP, which require square cluster sizes, is erratic. For example, SP class B runs significantly *slower* on 196 processors than on 144. In both cases, the actual allocated cluster size is 256 processors, so it is not clear why using fewer of the allocated processors results in better performance. Because our model does not capture the behavior of these programs, it may not be an appropriate choice for modeling a power-of-two machine.

The MG benchmark exhibits another behavior that our model does not capture: superlinear speedup. Many programs perform poorly on small cluster sizes because the performance of the memory system degrades as the amount of data per processor increases. For these programs, the speedup curve exceeds the theoretical upper bound of our model.

Fortunately, we can detect this behavior easily: a negative estimated σ indicates superlinear speedup. Thus, our model can be used to set a lower bound on the cluster size for a job: if σ is negative, we discard observations from the low end of the range until it is positive. It is probably not desireable to allocate a smaller cluster size, since the job would run inefficiently.

There is one other behavior that our model does not capture: non-monotonic speedup curves. On large cluster sizes, the communication overhead for some programs eventually overwhelms the benefits of additional parallelism and the speedup curve begins to decline. Of course, there is little value in modeling this behavior, since it is never desireable for an job to allocate a large cluster if it runs faster on a smaller one. If we observe declining speedups, we can discard observations from the upper end of the range until the curve is monotonic, and impose an upper bound on the cluster size that many be allocated.

4.2 SPLASH-2 programs

We obtained speedup curves for the SPLASH-2 programs running on a simulated shared-memory computer [19] [18]. The Stanford Parallel Applications for Shared memory (SPLASH) suite consists of 8 complete programs and 4 computation kernels that are intended to span a wide range of scientific applications. These include LU decomposition, a ray-tracing program, an ocean model, an n-body solver, and more. For details, see http://www-flash.stanford.edu.

For each program, we obtained the measured speedup on 6 cluster sizes (2, 4, 8, 16, 32 and 64 processors). The speedup on one processor is defined to be one. Figure 7 shows these observed speedups and the speedup model we estimated for each program. In each case, we observe that the fitted model is a good match for the observed data.

Each graph is labeled with the name of the benchmark and the estimated parameters A and σ. Two of the programs exhibit nearly linear speedup. Others have much more limited parallelism; the lowest value is A is 20.3. The value of σ is generally less than 1, although two programs yielded estimates of $\sigma = 1.7$ and $\sigma = 2.7$. The distribution of σ is similar on the NAS benchmarks.

5 Conclusions

- Our proposed speedup model captures the behavior of numerous scientific applications running on a variety of parallel computers, both shared- and distributed-memory. Thus, we feel that this model is a realistic choice for modeling parallel workloads.
- The parameters of our model correspond to measureable program charactericstics. Thus, our observations of these benchmarks give us some insight into the values and distibutions of these parameters in a real workload.

5.1 Future work

We have suggested that we can infer the program characteristics A and σ of a program by fitting our model to observed speedups. We have shown that our model fits observed speedups well; thus it would be appropriate to use these estimated parameters for allocation and scheduling. But we have not demonstrated that the estimated parameters necessarily reflect the *actual program characteristics* as they might be derived from a known parallelism profile. We have identified cases in which they do not, and these cases suggest restrictions on when and how this approach will be successful. In future work, we plan to clarify these restrictions and determine how meaningful the estimated parameters really are.

References

[1] Timothy B. Brecht and Kaushik Guha. Using parallel program characteristics in dynamic processor allocation policies. In *Proceedings of Performance '96*, October 1996.

[2] Soumen Chakrabarti, James Demmel, and Katherine Yelick. Modeling the benefits of mixed data and task parallelism. In *Seventh Annual ACM Symposium on*

Parallel Algorithms and Architectures (SPAA '95), July 1995.

[3] Su-Hui Chiang, Rajesh K. Mansharamani, and Mary K. Vernon. Use of application characteristics and limited preemption for run-to-completion parallel processor scheduling policies. In *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, 1994.

[4] Lawrence W. Dowdy. On the partitioning of multiprocessor systems. Technical Report 88-06, Vanderbilt University, March 1988.

[5] Allen B. Downey. A parallel workload model and its implications for processor allocation. Technical Report CSD-96-922, University of California at Berkeley, 1996.

[6] Allen B. Downey. Using queue time predictions for processor allocation. Technical Report CSD-97-929, University of California at Berkeley, 1997.

[7] Derek L. Eager, John Zahorjan, and Edward L. Lazowska. Speedup versus efficiency in parallel systems. *IEEE Transactions on Computers*, 38(3):408–423, March 1989.

[8] Dipak Ghosal, Giuseppe Serazzi, and Satish K. Tripathi. The processor working set and its use in scheduling multiprocessor systems. *IEEE Transactions on Software Engineering*, 17(5):443–453, May 1991.

[9] Kaushik Guha. Using parallel program characteristics in dynamic multiprocessor allocation policies. Technical Report CS-95-03, York University, May 1995.

[10] Cathy McCann and John Zahorjan. Scheduling memory constrined jobs on distributed memory parallel computers. Technical Report UW-CSE-94-10-05, University of Washington, 1994.

[11] William H. Press, Brian P. Flannery, Saul A Teukolsky, and Willian T. Vetterling. *Numerical Recipes in C.* Cambridge University Press, Cambridge, MA, 1988.

[12] Emilia Rosti, Evgenia Smirni, Giuseppe Serazzi, and Lawrence W. Dowdy. Analysis of non-work-conserving processor partitioning policies. In *IPPS '95 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 101–111, 1995.

[13] Willian Saphir, Alex Woo, and Maurice Yarrow. The NAS parallel benchmarks 2.1 results. Technical Report NAS-96-010, Numerical Aerospace Simulation Facility, NASA Ames Research Center, 1996.

[14] Sanjeev. K Setia. The interaction between memory allocation and adaptive partitioning in message-passing multicomputers. In *IPPS '95 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 89–99, 1995.

[15] Sanjeev K. Setia and Satish K. Tripathi. A comparative analysis of static processor partitioning policies for parallel computers. In *Proceedings of the Internationsal Workshop on Modeling and Simulation of Computer and Telecommunications Systems (MASCOTS)*, January 1993.

[16] Kenneth C. Sevcik. Characterizations of parallelism in applications and their use in scheduling. *Performance Evaluation Review*, 17(1):171–180, May 1989.

[17] Evgenia Smirni, Emilia Rosti, Lawrence W. Dowdy, and Giuseppe Serazzi. Evaluation of multiprocessor allocation policies. Technical report, Vanderbilt University, 1993.

[18] Steven C. Woo. Personal correspondence. 1996.

[19] Steven C. Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *22nd Annual International Symposium on Computer Architecture*, pages 24–36, June 1995.

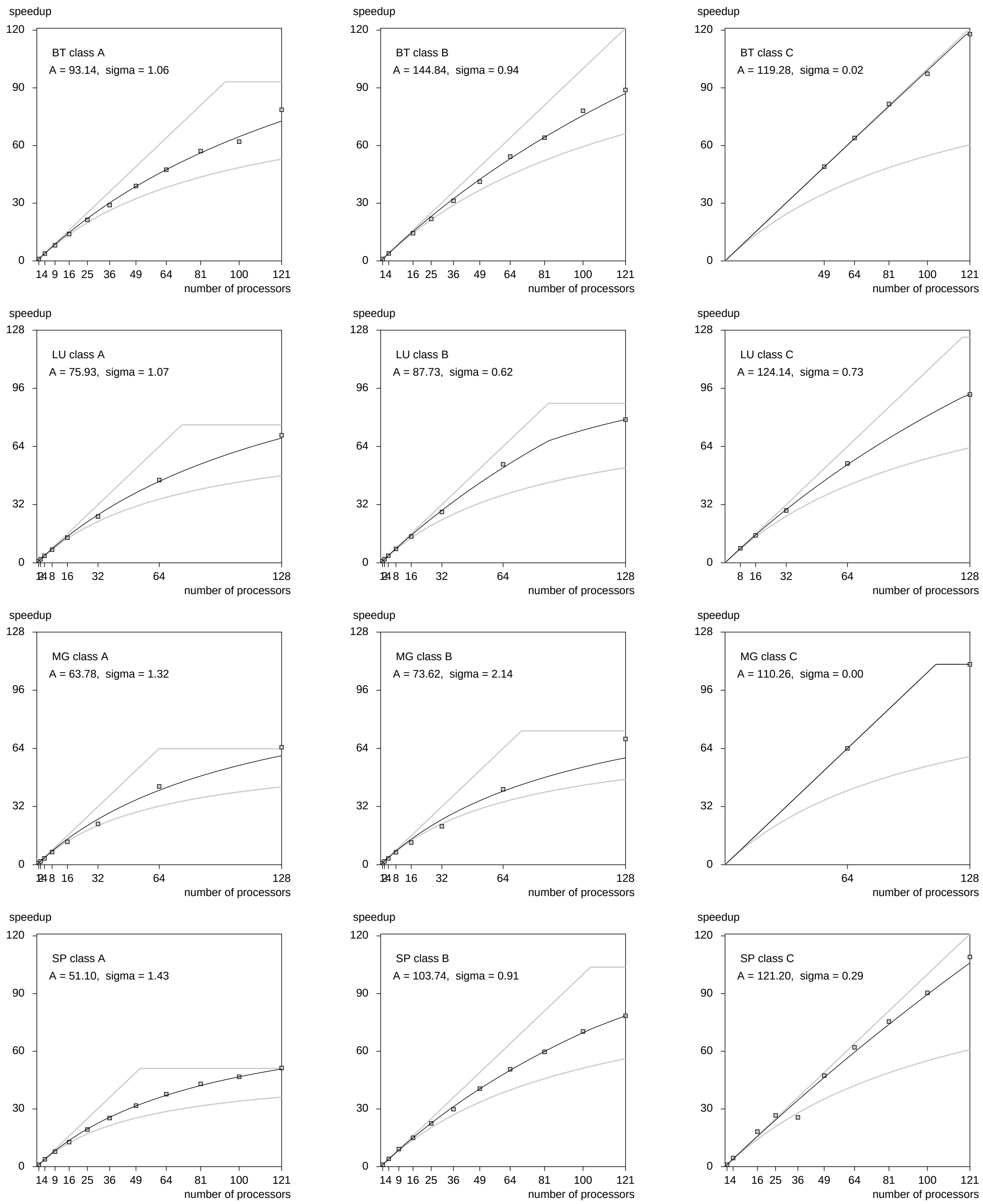


Figure 5: Speedup curves for the NAS benchmarks run on the IBM SP2 at NAS. Each row reports the results of one benchmark on three difference problem sizes. Class A is the smallest problem size; Class C is the largest. The gray lines show the theoretical upper limit ($\sigma = 0$) and lower limit ($\sigma = \infty$) for a program with the given average parallelism, A.

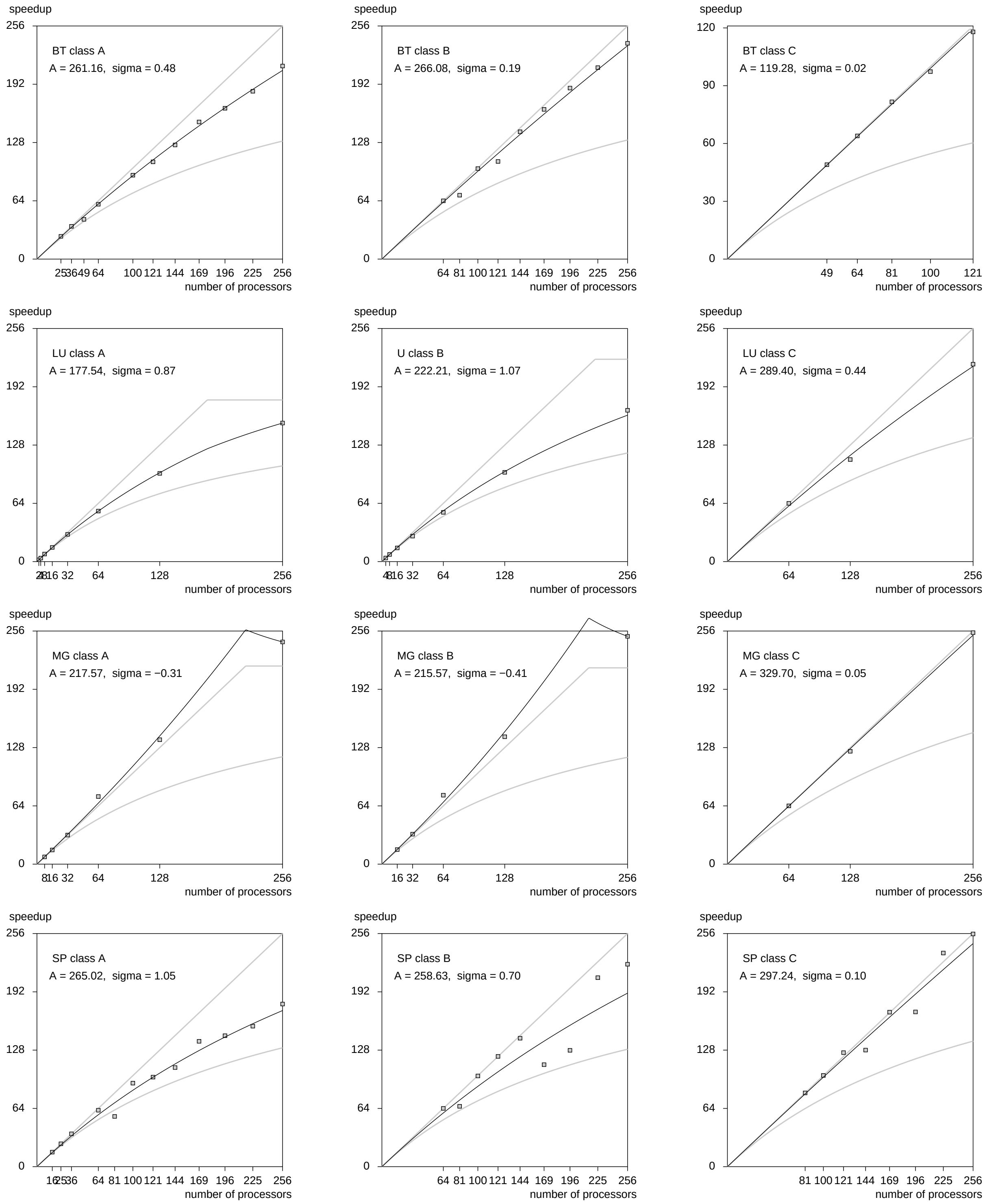


Figure 6: Speedup curves for the NAS benchmarks run on the Cray T3D at NAS. Each row reports the results of one benchmark on three difference problem sizes. Class A is the smallest problem size; Class C is the largest.

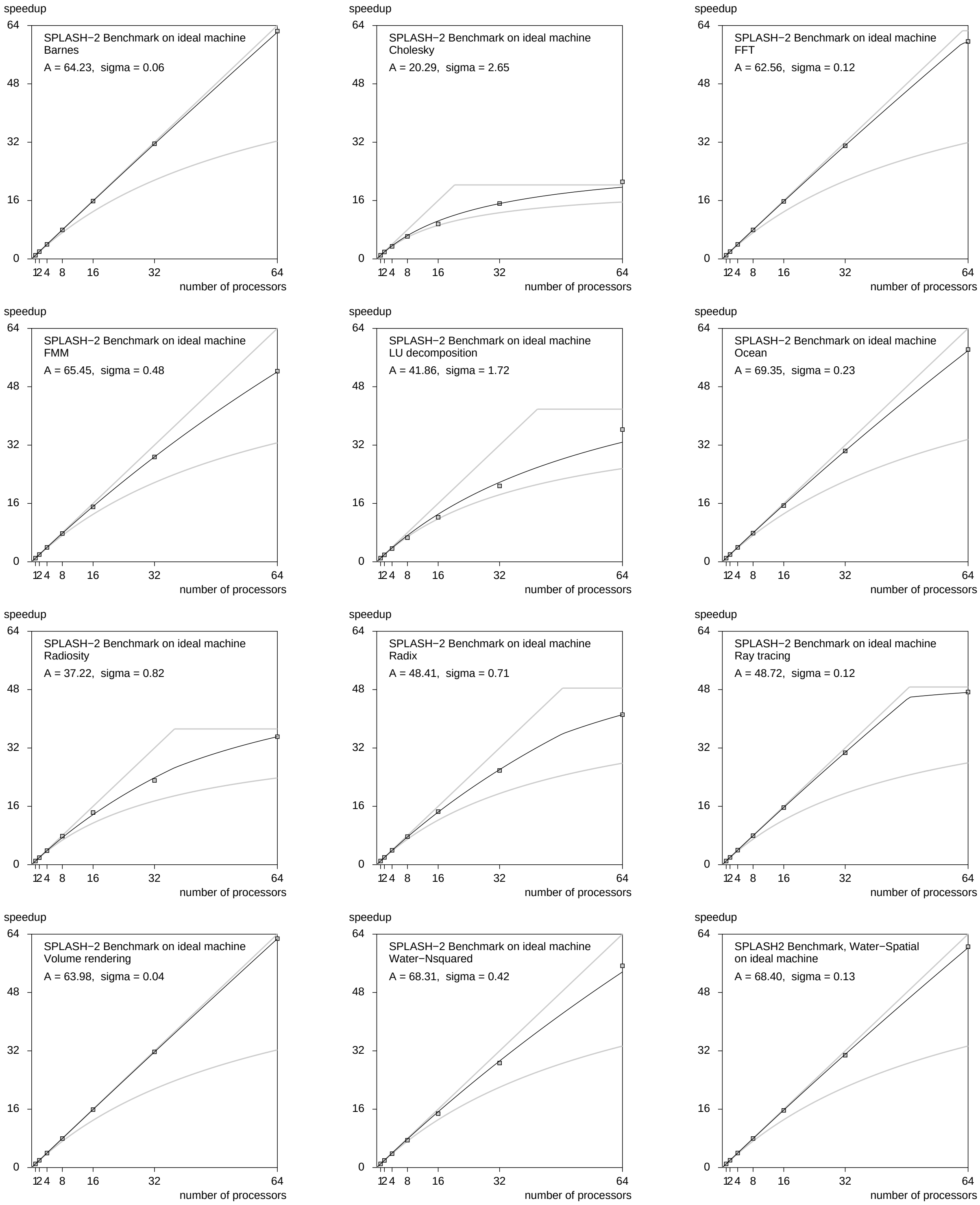


Figure 7: Speedup curves for the SPLASH-2 benchmark run on a simulated shared-memory machine.