Mathematical Models for the Speedup of Parallel Material Dynamics Codes

Darrell L. Hicks
Ken Kennedy
Lorie M. Liebrock

CRPC-TR93343
November 1993

Center for Research on Parallel Computing
Rice University
P.O. Box 1892
Houston, TX 77251-1892

Abstract

There have been some previous predictions of possible parallel speedups which seem to be overly pessimistic in their prognostications for the advantages to be gained by parallel processing in numerical material dynamics. There seems to be no insurmountable sublinear limit to the growth of the speedup in certain carefully designed parallel material dynamics algorithms for parallel machines with appropriate architectures.

1 Introduction

The origin of the basic idea of parallelization speedup is lost in prehistorical antiquity. Current conventional wisdom and recent opinion has been influenced by Amdahl(1967), Lee(1977,1980), Buzbee(1983), Flatt(1984), Larson(1984), Hwang & Briggs(1984). The producers of this paper have also been prone to proliferating the prose on parallelization paradigms [4, 8, 18, 19, 20, 21].

Examples of material (solid, liquid, gas, plasma, multiphase) dynamics applications and background references may be found in reference [9]. Computational experiments with a variety of parallel algorithms for material dynamics on a variety of parallel machines [5, 7, 8, 10, 11, 12, 13, 18, 19, 20, 21] suggest the speedup models presented here. In particular, some results on parallelized versions of the von Neumann-Richtmyer(vNR) scheme[6] will be discussed.

2 Speedup Models

Let $T(n)$ be the run time of a parallel computation using n processors. We have considered timing models of the following form

$$T(n) = T_s + T_p/n + T_O(n) \tag{1}$$

where the parallelization overhead for n processors, $T_O(n)$, satisfies $T_O(1) = 0$ and thus $T(1) = T_s + T_p$; T_s is due to the part of the program which must be run in serial; T_p is due to the part of the program which may be divided into n equal subparts and run in parallel. Typically it is assumed that the

model for the overhead time, $T_O(n)$, is unboundedly increasing with n; *e.g.*, it is often assumed that $T_O(n)$ has linear or at least logarithmic growth with n. We refer to (1) as the T_{spO} **model.**

We have used the T_{spO} model along with empirical data from our computational experiments to predict the limits of the growth of such architectures as found in the HEP, ELXSI, Cray, Intel iPSC, et cetera on algorithms such as our parallelized vNR [4, 8, 18, 10, 11, 12, 13, 18, 19, 20, 21]. We also have used the T_{spO} model to study the marginal gain efficiency and the number of processors to use for optimal cost effectiveness[4, 18].

Caveats: We have the following caveats to pronounce on the T_{spO} model, related models and their prognostications.

First, although it is not necessarily obvious, one can get into an **ill-conditioned calculation** in attempting to fit the T_{spO} model to timing data[18]. Thus the results of such exercises must be taken cum grano salis.

Second, the amount of parallelization overhead is strongly influenced by the nature of the dependences in the parallel algorithm. For example, distributed memory machines with mesh topologies, when used for certain material dynamics calculations having only nearest neighbor mesh topology dependences, appear to have an effective parallelization overhead which becomes negligibly small as the granularity size becomes sufficiently large. Further, the effective parallelization overhead for these calculations does not appear to be an unboundedly increasing function of the number of processors.

Finally, the T_{spO} model, as given in (1), does not fit the natural model for certain parallel algorithms for material dynamics.

To illustrate, let us take the case of a parallel calculation where both the algorithm and the machine have the topology of a one-dimensional mesh. In particular we will consider a numerical material dynamics problem.

Let each processor have z zones to advance computationally. Suppose it takes each processor the same time, t_z, to advance z zones one computational cycle (e.g., a timestep). Further, suppose the amount of time that processor p, $1 \leq p \leq n$, takes to complete communications with its left and right neighbors on one cycle and/or compute boundary values is τ_p. (We call the values communicated in this example "internal boundary values" for the obvious reason.) Then, the total time for processor p to advance z zones by one cycle, communicate with its neighbors and/or compute boundary values, and be prepared to start the next computational cycle is $t_z + \tau_p$. Let $\tau_{\max}(n) = \max_{1 \leq p \leq n} \tau_p$; $\tau_{\max}(n)$ is **not** necessarily increasing unboundedly

as n increases. There are, of course, communication protocols which would make $\tau_{\max}(n)$ increase unboundedly with n. However, there are also simple communication protocols which do not have $\tau_{\max}(n)$ increasing unboundedly with n.

Consider a parallel material dynamics algorithm with the following steps in its computation-communication cycle. The communications are all done in asynchronous mode. That is, the transmissions may be thought of as asynchronous, buffered transfers from the sending processor's local memory to the receiving processor's local memory. The leftmost (rightmost) processor has index $p = 1 (p = n)$, does not send internal boundary values to the left (right) and gets its left (right) side information by evaluating the left (right) boundary value functions.

The steps for each computational cycle are:

1. Compute (includes fetchs from and stores to local memory) for time t_z.
2. Transmit internal boundary values leftward (except $p = 1$).
3. Transmit internal boundary values rightward (except $p = n$).
4. If the number of computational cycles desired is completed, then stop; else, go to (1).

Let $\tau_{lb}(\tau_{rb})$ be the time it takes to compute the left (right) boundary values; $\tau_{tl}(\tau_{tr})$ be the time to transmit the internal boundary values to the left (right), then

$$\tau_{\max}(1) = \tau_{lb} + \tau_{rb} \ ;$$

$$\tau_{\max}(2) = \max(\tau_{lb} + \tau_{tr} \ , \ \tau_{tl} + \tau_{rb}) \ ;$$

$$\tau_{\max}(3) = \max(\tau_{lb} + \tau_{tr} \ , \ \tau_{tl} + \tau_{tr} \ , \ \tau_{tl} + \tau_{rb})$$

Note that, for $n \geq 3$, $\tau_{\max}(n)$ is not increasing with n, it is a constant, i.e.,

$$\tau_{\max}(n) = \tau_{\max}(3)$$

for $n \geq 3$.

Remarks: $\tau_{lb} + \tau_{rb}$, i.e., $\tau_{\max}(1)$, is, in our experience, usually (although not always) negligible. Observe that τ_{tl} and τ_{tr} should be, for greater precision, defined as the excesses of the times to transmit the internal boundary

values to neighboring memories over the times to store to local memory. Note that if it happened by chance that $\tau_{lb} = \tau_{rb} > \tau_{tl} = \tau_{tr}$, then we could have the amusing situation $\tau_{\max}(1) > \tau_{\max}(2) = ... = \tau_{\max}(n)$, for $n \geq 2$; although possible this case does not seem probable. It appears, from our experiences, that the more probable case is $\tau_{\max}(1) \leq \tau_{\max}(2) \leq \tau_{\max}(3) = ... = \tau_{\max}(n)$, for $n \geq 3$; our test problems for the parallel vNR codes are in the latter case.

For such calculations, we submit that a reasonable model for the time, $T(n, z)$, to calculate z zones with $n \geq 1$ processors is

$$T(n, z) = t_z + \tau_{\max}(n) \tag{2}$$

Then, a reasonable model for the **speedup,** $S(n, z)$, with n processors each working on z zones is $T(1, nz)/\ T(n, z)$, i.e., using $T(1, nz) = nt_z + \tau_{\max}(1)$ we have

$$S(n, z) = \frac{nt_z + \tau_{\max}(1)}{t_z + \tau_{\max}(n)} \tag{3}$$

This has a **parallelization efficiency,** $E(n, z)$, of

$$E(n, z) = \frac{1 + \tau_{\max}(1)/(nt_z)}{1 + \tau_{\max}(n)/\ t_z} \tag{4}$$

To emphasize: The point we wish to make is that for certain calculations, where the problem topology matches the machine topology, the parallelization overhead is not only not necessarily unboundedly increasing with the number of processors but is negligible when $\tau_{\max}(n)/\ t_z$ is negligible. The ratio $\tau_{\max}(n)/\ t_z$ is, of course, the communication/computation whose importance to efficient parallelization is well known:

$$C_{CR}(n, z) = \tau_{\max}(n)/t_z$$

Recall that z is the number of zones per processor. It is straightforward to show that $t_z = O(z)$. Further, for one, two, and three dimensional problems $\tau_{\max}(n)/\ t_z = O(z^{-1})$, $O(z^{-1/2})$, and $O(z^{-2/3})$, respectively, when we assume square, and cubic geometry for the two, and three dimensional problems.

Observe that the communication/computation ratio $C_{CR}(n, z) = O(z^{-q})$, where $q = 1,\ 1/2$, and $2/3$ for one, two, and three dimensional problems, may be made as small as we like by increasing z;(i.e., increasing the granularity

size). This illustrates, one more time, one of the well-known advantages of large granularity parallelism.

We encounter difficulties if we attempt to force the timing model in (2) into the T_{spO} mold. First, observe that the T_s term was neglected in the derivation of (2) (we discuss the negligibility of T_s later); i.e., $T_s = 0$. Our intuitive impulse is to relate $\tau_{\max}(n)$ to $T_O(n)$. However, $T_O(1) = 0$ is required by the T_{spO} model and $\tau_{\max}(1) = \tau_{lb} + \tau_{rb} \neq 0$. To fix this we might try defining $T_O(n) = \tau_{\max}(n) - \tau_{\max}(1)$ but that would force us into the contradiction $T_s = \tau_{\max}(1) = \tau_{lb} + \tau_{rb} \neq 0$; this is nonsense from several points of view; for one thing, the left and right boundary conditions need not be done in serial, they could be done in parallel in most cases of interest. Therefore, we see that the (2) model, the natural model (in our opinion) for the time of the parallel algorithms for material dynamics we have designed and described, does not fit the T_{spO} mold.

Thus, our experiences with parallel algorithms for material dynamics suggest that (1) should be generalized to

$$T(n, z) = T_s(n, z) + T_p(n, z)/n + T_{OC}(n, z) \tag{5}$$

where the $T_{OC}(n, z)$ term is **not** required to go to zero at $n = 1$. The $T_{OC}(n, z)$ term contains times due to parallelization overhead **plus** other computation and communication times which do **not** fit into the $T_s(n, z)$ term (which contains all of the serial time) **nor** into the $T_p(n, z)/n$ term (which contains the n-equal-subparts-parallelizable computation time). We call this the T_{spOC} **model.**

To relate (5), the T_{spOC} model, to (2): we have neglected the T_s term in (2), i.e., $T_s(n, z)$ is zero; $T_p(n, z)/n$ is t_z; $T_{OC}(n, z)$ is $\tau_{\max}(n)$.

Equation (5) may be considered in several different ways:

(i) with n free, i.e., the job size is considered to be growing linearly with n,

(ii) with the nz product held fixed, i.e., job size, J_S, is a constant.

In case (ii) it is convenient to use the notation $t_z = r_{cp}z$ where r_{cp} is the time to compute one zone for one cycle. Thus, if $nz = J_S$ then, $z = J_S/\ n$ and $t_z = zr_{cp}$. This yields a model for **divide-the-job speedup** (which is commonly simply called the speedup).

Case (i) yields a model for **multiply-the-job speedup**.

Now assume we are given a specific, fixed machine. When J_S (measured in the number of zones in the job) is the size of the largest job that can fit on a single processor, then we call the resulting speedup with all n processors (each working on $z = J_S/\ n$ zones) the **practical divide-the-job speedup.** When $J_S * n$ is the size of the largest job that can fit on all n processors (each processor working on $z = J_S$ zones) of that machine, then we call the resulting speedup the **practical multiply-the-job speedup.** To abbreviate the prose, we suppress the adjective "practical" when it is, readily understood from context.

Note that the multiply-the-job speedup has an advantage over the divide-the-job speedup for a particular parallel processor with a fixed number, n, of processors due to the lower communication/computation ratio. That is, $C_{CR}(n,z) = \tau_{\max}(n)/t_z$ for $n \geq 3$ is reduced by a factor of $1/n$ in the multiply-the-job speedup because multiply-the-job speedup allows us to increase the granularity size by a factor of n over the granularity size in the divide-the-job speedup.

Measuring multiply-the-job speedups: In practice, it may appear to be difficult, if not impossible, to measure the multiply-the-job speedups (see section 4 for further explanation of the apparent difficulty). However, we can approximate the multiply-the-job speedups by ratios of computation rates which are readily measurable. We now justify this approximation. Let $C_R(n,z)$ be the computation rate (measured in zones computed per second) for $n \geq 1$ processors each computing z zones; then

$$C_R(n,z) = \frac{nz}{t_z + \tau_{\max}(n)} \tag{6}$$

Therefore, the computation rate ratio is

$$C_R(n,z)/\ C_R(1,z) = \frac{n[t_z + \tau_{\max}(1)]}{t_z + \tau_{\max}(n)} \tag{7}$$

Note that

$$C_R(n,z)/\ C_R(1,z) \geq S(n,z) \tag{8}$$

Further note that if $\tau_{\max}(1)$ is negligible, then

$$C_R(n,z)/\ C_R(1,z) \doteq S(n,z) \tag{9}$$

where $S(n,z)$ is given by (3). Thus the computational rate ratio and running time ratio have approximately the same recipes when $\tau_{lb} + \tau_{rb}$ is negligible.

Hence the computational rate ratio may be used to estimate the running time ratio when $\tau_{\max}(1)$ is negligibly small as it is in our test problems for the vNR algorithms.

Remark: Observe that we could compute $S(n,z)$ indirectly from quantities we can measure as follows. Given n and z, measure t_z, $\tau_{\max}(1)$, and $C_R(n,z)$ with timing routines, then use (6) to compute $\tau_{\max}(n)$, and then put the values of t_z, $\tau_{\max}(1)$, and $\tau_{\max}(n)$ into (3) to compute $S(n,z)$. Further note that if we are given the computation and communication rates of a particular parallel processor then we can a priori approximate all the terms in $S(n,z)$ in (3). That is, the values of t_z and $\tau_{\max}(n)$ for $n = 1,2,3$ are not difficult to figure out by doing some operation counting; and then $\tau_{\max}(3) = \tau_{\max}(n)$ for $n \geq 3$. The last equality suggests that computational experiments for $n > 3$ are not necessary on appropriate architectures for predicting the possible speedups of parallel algorithms obeying (3).

3 The T_s Term

Recall that in deriving (3) we neglected the T_s term and remarked that we would discuss this at a later time. "The time has come, the Walrus said ..."

What is wrong with the speedup analysis we presented in section 2? Well, for one thing we completely neglected input and output. We submit, with tongue only slightly in cheek, that this oversight is all too common. The initial data involves $O(nz)$ numbers that need to be read into the machines to start the computation. Likewise, complete output of all information at any timestep also involves $O(nz)$ numbers.

Assume the input/output device is a single serial machine. The best time we know for moving $O(nz)$ arbitrary, distinct numbers into (or out of) a single, serial machines from (or to) $n \geq 1$ machine(s) is $O(nz)$. This is because just getting $O(nz)$ numbers into (or out of) a single serial machine takes $O(nz)$ time; in other words, we have hypothesized a "serial bottleneck" for input and output.

Of course, we could consider parallel input and output but, for now, let us see what a serial bottleneck for input and output does to our previous results.

Let $T_{io}(n,z)$ be the time taken for input and output of a total of nz zones on n processors (each processor having z zones). For simplicity, suppose that

input occurs only at the beginning and output occurs only at the end of the calculation. This does not seem to be an unreasonable assumption when one considers that in a well designed machine the intermediate input and output can be buffered in and buffered out. Let the input plus output time per zone be r_{io} and assume the following simple model for input/output time

$$T_{io}(n,z) = nzr_{io} \tag{10}$$

Let $T_\nu(n,z)$ be the total time for a run with ν computational cycles on n processors with z zones each, i.e.,

$$T_\nu(n,z) = T_{io}(n,z) + \nu T(n,z) \tag{11}$$

where $T(n,z)$ is the computational time for one cycle given by (2). Thus we have for $n \geq 1$

$$T_\nu(n,z) = nzr_{io} + \nu[t_z + \tau_{\max}(n)] \tag{12}$$

Remark: To relate (12) to the T_{spOC} model: $T_s(n,z)$ is nzr_{io}; $T_p(n,z)/n$ is νt_z; $T_{OC}(n,z)$ is $\nu\tau_{\max}(n)$.

From (12), the multiply-the-job speedup is

$$S_\nu(n,z) = \frac{n\{\nu[t_z + \tau_{\max}(1)/\ n] + zr_{io}\}}{\nu[t_z + \tau_{\max}(n)] + nzr_{io}} \tag{13}$$

and the parallelization efficiency is

$$E_\nu(n,z) = \frac{1 + \tau_{\max}(1)/(nt_z) + zr_{io}/(\nu t_z)}{1 + \tau_{\max}(n)/\ t_z + nzr_{io}/(\nu t_z)} \tag{14}$$

In a typical explicit code (such as those based on the original vNR), we have a CFL-type (Courant-Friedrichs-Lewy) timestep restriction which restricts stress-wave propagation distances to at most one zone per cycle[6]. It is typical in stress-wave propagation calculations to allow waves to make at least one transit of the material (or spatial) interval and more often the waves make multiple (say m) transits. In this case we have $\nu = mnz$ and thus (14) becomes

$$E_\nu(n,z) = \frac{1 + \tau_{\max}(1)/(nt_z) + r_{io}/(mnt_z)}{1 + \tau_{\max}(n)/\ t_z + r_{io}/(mt_z)} \tag{15}$$

As m gets large (e.g., if an approach to thermodynamic-equilibrium in a material dynamics calculation is desired), (15) goes to

$$E_\nu(n,z) \approx \frac{1+\tau_{\max}(1)/(nt_z)}{1+\tau_{\max}(n)/\ t_z} \tag{16}$$

This is the same result as we got before (compare (16) with (4)) when we neglected the T_s term.

In the usual implementation of the original vNR algorithm, which involves an explicit discretization of the conservation laws of momentum and volume, there is a calculation for a stable timestep based on a constraint similar to the CFL inequality[6]. This leads to a global dependence, i.e., the timestep depends on quantities in every zone of the calculation and consequently requires information which must be gathered from every processor involved in the calculation. This is another potential contribution to T_s. However, we can do an a priori analysis and find a priori bounds on the quantities involved in the timestep constraint. Therefore, we can find a priori a timestep which is stable for the entire calculation. This modification of the usual implementation of the vNR scheme we call the APT (A Priori Timestep) modification[12]. Thus the APT modification obviates the explicit timestep computation's contribution to T_s.

Implicit versions of vNR also avoid a timestep calculation at the expense of introducing other surmountable parallelization difficulties[12].

4 Summary and Conclusions

The multiply-the-job speedup may be estimated by the computational rate ratio as shown in (9) or computed by one of the procedures described in the remark following (9). In general, it is impossible to measure directly because of storage limitations. That is, $T(1,nz)$ is impossible to measure directly when z is the maximum number of zones that can be loaded onto a single processor and $n \geq 2$. Perhaps this at least partially explains the prevalent preoccupation with the divide-the-job speedups.

In some computational experiments with a parallel, implicit vNR algorithm on the Intel iPSC[12] the 99% parallelization efficiency achieved in the multiply-the-job mode yields the result that the communication/computation

ratio was approximately given by

$$\tau_{\max}(32)/\ t_z \doteq 0.01$$

In other symbols, $\tau_{\max}(32)$ was about 1% of t_z in the iPSC calculations of the implicit vNR code with $z = 1999$ and $n = 32$. If the iPSC had allowed more zones per node, then we could have done even better than 99% parallelization efficiency in the multiply-the-job mode for arbitrarily large values of n.

Therefore, we see no insurmountable sublinear limits to the speedup of these parallel algorithms for material dynamics. That is, the speedup seems to grow as $\theta(z)\ n$ where $0 < \theta(z) < 1$ and $\theta(z)$ approaches unity as z (the size of the granularity) grows.

Underlying these results is the fact that our parallel algorithms' design is based on the principle of reducing all dependences to **nearest neighbor dependences**. This is not an insurmountable problem when developing parallel algorithms for simulating the dynamics of physical systems if the simulation is based on mathematical models with finite propagation speeds.

References

[1] G. M. Amdahl, Validity of the single processor approach to achieving large scale computer capabilities, *Proc. AFIPS Spring Joint Comp. Conf. 30*:483-485, Atlantic City, NJ, 1967.

[2] B. L. Buzbee, The efficiency of parallel processing, *Los Alamos Science* #9, 1983.

[3] H. P. Flatt(1984), A simple model for parallel processing, *Computer* 17(11):75, 1984.

[4] H. P. Flatt and K. Kennedy, Performance of parallel processors, *Parallel Computing* 12(1):1-20, 1989.

[5] K. E. Fletcher, C. A. Foulston, and L. M. Liebrock, A feasibility study of portable parallel programming using parallel language subroutines, term paper in Distributed Systems Course in Department of Computer Science at Rice University, Fall Semester 1989.

[6] D. L. Hicks, Stability analysis of WONDY(a hydrocode based on the artificial viscosity method of von Neumann and Richtmyer) for a special case of Maxwell's material law, *Mathematics of Computation* 32(144):1123-1130, 1978.

[7] D. L. Hicks, Parallel processing algorithms for hydrocodes on a computer with MIMD architecture(Denelcor's HEP), *Idaho National Engineering Labs.*, EGG-SAAM-6452, 1983.

[8] D. L. Hicks, Hydrocodes on the HEP, In J. S. Kowalik, editor ,*MIMD Computation: HEP Supercomputer and its Applications*, pages 309-330. M.I.T. Press, 1985.

[9] D. L. Hicks, Introduction to computational continuum dynamics: A personal perspective, *Applied Mathematics and Computation* 20(1):5-19, 1986.

[10] D. L. Hicks and L. M. Liebrock, Parallel algorithms for hydrocodes, *Transactions of the American Nuclear Society* 55:330-331, 1987.

[11] D. L. Hicks, L. M. Liebrock, V. A. Mousseau and G. A. Mortensen, Parallelization of hydrocodes on the Intel hypercube, part 1, *Idaho National Engineering Laboratories*, EGG-SPAG-7682, 1987.

[12] D. L. Hicks, L. M. Liebrock, V. A. Mousseau and G. A. Mortensen, Parallelization of hydrocodes on the Intel hypercube, part 2", *Idaho National Engineering Laboratories*, EGG-SPAG-7818, 1987.

[13] D. L. Hicks, J. F. McGrath and L. M. Liebrock, "Thermal-hydraulics algorithms for parallel processors", *Transactions of the American Nuclear Society*, 50:275-276, 1985.

[14] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill, 1984.

[15] J. L. Larson, Multitasking on the Cray X/MP-2 multiprocessor, *Computer* 17(7):62-69, 1984.

[16] R. B. L. Lee, Performance bounds in parallel processor organizations, In D. J. Kuck, D. Lawrie, and A. Sameh, editors, *High Speed Computer and Algorithm Organization*, Academic Press, 1977.

[17] R. B. L. Lee, Empirical results on the speed, efficiency, redundancy, and quality of parallel computations, *International Conf. Parallel Proc.* pages 91-96, 1980.

[18] L. M. Liebrock, J. F. McGrath and D. L. Hicks, Experiments in concurrent computational dynamics on the CRAY X-MP, ELXSI and HEP parallel processors with recommendations for parallel architectures & languages", *KMS Fusion Inc.*, KMSF-U-1784, 1986.

[19] J. F. McGrath, D. L. Hicks and L. M. Liebrock, Parallel processing for computational continuum dynamics, In S. Lakshmivarahan, editor, *Parallel Processing using the Heterogeneous Element Processor*, University of Oklahoma Press, pages 77-96, 1985.

[20] J. F. McGrath, D. L. Hicks and L. M. Liebrock(1985b), Parallel algorithms for computational fluid dynamics", (final report on a project supported by AFOSR under contract # F49620-84-6-0111), *KMS Fusion Inc.*, KMSF-U1601, 1985.

[21] J. F. McGrath, D. L. Hicks and L. M. Liebrock, Concurrent algorithms for computational continuum dynamics, *Applied Mathematics and Computation* 20(2):145-173, 1986.