Adaptation of Asymptotic Close-Coupling Methods to Massively Parallel Computers

T. W. Gorczyca and M. S. Pindzola
Department of Physics, Auburn University, Auburn, AL 36849

F. S. Shieh and C. L. McCreary
Department of Computer Science, Auburn University, Auburn, AL 36849
(February 21, 1995)

Abstract

A standard asymptotic close-coupling method based on a perturbation technique is adapted for use on massively parallel computers. The calculation of K-matrices and cross sections is partitioned over the processors according to incident electron energy. Use of external disk during the calculation is eliminated. Linear scaling with the number of processors is achieved for a 128-node Intel iPSC/860 and a small farm of Sun Sparc 10 workstations.

I. INTRODUCTION

The R-matrix method [1] remains one of the most successful techniques for solving the close-coupled scattering equations associated with electron-atom collisions. A sphere of radius equal to the physical extent of the atom divides space into an inner and outer region. Inside the sphere ($r \leq a$) the solution of the Schrödinger equation for the scattered electron and the N-electron target atom or ion is found by diagonalization of an (N+1)-electron Hamiltonian matrix constructed from a basis set of bound and continuum functions. The continuum basis functions, which represent the motion of the scattered electron, are non-zero on the surface of the sphere. From the surface amplitudes, $w_{i\alpha}$, and eigenvalues, ϵ_α, an R-matrix is obtained for all energies, E, and each set of conserved quantum numbers given by the expression

$$R_{ij}(E) = \frac{1}{2a} \sum_\alpha \frac{w_{i\alpha} w_{j\alpha}}{\epsilon_\alpha - E} , \tag{1}$$

The dimension of the basis function expansion parameter, α, is typically 1-2 orders of magnitude larger than the dimension of the channel indices i or j.

Outside the sphere the solution of the Schrödinger equation for the electron-atom collision yields a set of coupled differential equations given by (in atomic units):

$$\left(-\frac{1}{2}\frac{d^2}{dr^2} + \frac{\ell_i(\ell_i+1)}{2r^2} + V_{ii}(r) - \frac{k_i^2}{2} \right) F_{ik}(r) + \sum_{j \neq i}^{n} V_{ij}(r) F_{jk}(r) = 0 , \tag{2}$$

where $F(r)$ is the radial wavefunction, ℓ_i is the orbital angular momentum quantum number, and k_i is the linear momentum of the scattered electron. The potential V_{ij} is given by:

$$V_{ij}(r) = \sum_{\lambda=0}^{\lambda_m} \frac{C_{ij}^{(\lambda)}}{r^{\lambda+1}} \ , \tag{3}$$

where the $C_{ij}^{(\lambda)}$ are long-range potential coefficients. The coupled equations of Eq.(1) are integrated outwards for each incident electron energy ($\frac{k_i^2}{2}$), subject to the R-matrix boundary conditions at the surface of the sphere, to yield K-matrices and thus collision cross sections.

For the photoionization and electron-impact excitation of atomic ions a suite of computer codes based on the R-matrix method has been developed at Queen's University of Belfast [2]. The asymptotic close-coupling equations of Eq.(1) are solved in a computer code called STGF using a perturbation technique due to Seaton [3]. To map out the detailed resonance structure found in the excitation cross sections of complex atomic ions, the asymptotic equations of Eq.(1) may have to be solved at many thousands of incident electron energies. Thus a substantial portion of the total time needed to make an R-matrix calculation for an electron-ion cross section may be spent running the STGF code. In the paragraphs below we outline how the STGF code may be easily adapted for execution on a massively parallel computer with a concurrent large decrease in wall clock time. In the next section we describe the program STGF. Timing demands for parallel-processing machines are discussed in Section 3. We show the results of a simple test case in Section 4, where particulars of the various machines are addressed. Then the results of a large case are described in Section 5, followed by concluding remarks.

II. DESCRIPTION OF STGF PROGRAM

The principle functions of the program STGF [2] which concern the present discussion are shown in the solid-lined boxes in Fig. 1. Briefly, the goal is to compute the total collision strength, $\Omega_{ij}^{tot}(E)$, for excitation between target states i and j ($i, j = 1, ..., N_s$) at perhaps a large number of energy values, N_e. Since the Hamiltonian is separable into various symmetries, or partial waves, denoted by the spin, angular momentum, and parity quantum numbers, $SL\pi$, the total collision strength is just the sum of contributions from each partial wave, i.e.

$$\Omega_{ij}^{tot}(E) = \sum_{SL\pi} \Omega_{ij}^{SL\pi}(E) \ . \tag{4}$$

The program is organized such that the outer-most loop is indexed by the partial wave symmetries. For each partial wave, input data, mainly the $w_{i\alpha}$ and $C_{ij}^{(\lambda)}$ arrays, is read, and a loop over energy points is performed. For each energy, the partial wave contribution, $\Omega_{ij}^{SL\pi}(E)$, is computed and added to the running sum, $\Omega_{ij}^{tot}(E)$. In the normal mode of operation, this is accomplished by storing $\Omega_{ij}^{tot}(E)$ in a file on disk, which requires a read and write of this array for every energy and partial wave. Using a disk file is necessary on small-memory machines when both the number of indices of the Ω array (equal to $N_s(N_s+1)/2$) and the number of energies, N_e, are large. After all loops have been completed, this array is read from the disk file and written to a sequential-access output file.

The modifications to this program for parallel computers is as follows. First, the energy points at which the collision strength is to be evaluated are distributed among the number of available processors, N_p. On the iPSC/860, this number is a power of two, i.e. $N_p = 2^M$, where M is an integer (the cube size). We choose the number of total energy points, N_e^{tot}, to be divisible by 128, the maximum number of nodes. Thus, each node must compute the collision strength at a number of energy points $N_e = N_e^{tot}/N_p$. Usually these energies are given by a constant mesh $E_i = E_0 + i\Delta E,\ i = 0, ..., N_e - 1$. We distributed the energies such that node number zero is assigned the energies $\{E_0, E_{N_e}, ..., E_{(N_p-1)N_e}$, node number one is assigned $\{E_1, E_{N_e+1}, ..., E_{(N_p-1)N_e+1}$, etc. This interleaved partitioning scheme has been shown to yield good load balancing for a large number of energies [4]. For the present program, however, this is expected to be the most efficient partitioning scheme for any number of energies, as we will show when considering the computational timing in the next section.

The second main modification is that the READ and WRITE commands are only performed by one processor, since the iPSC/860 machine has this limitation. In the case of a READ, this processor, hereafter referred to as node number zero, then broadcasts the information out to all the other nodes. In the case of the final WRITE execution, all the other nodes first send their computed $\Omega_{ij}^{tot}(E)$ to the zeroth node, and this node then writes the Ω array to an output file.

The third modification concerns storage of the Ω array. As mentioned above, this array is normally stored in a disk file. We have eliminated this method of storage in favor of random-access memory storage, since otherwise the message passing between nodes in order to read from and write to the disk file at every energy would be extremely time consuming. Also, since writing data to a disk file is slower than writing to internal memory [5], a substantial savings is achieved by this change even for a single processor.

Using this method of storage, however, means that the Ω array must be stored in internal memory. The iPSC/860 machine we used for the following calculations had one megaword of memory per node. Typically more than half of that space is taken up by various other arrays within the program, which leaves space for an omega array of about 500,000 REAL*8 numbers. For a large calculation, the number of states might be about N_s=100, so that each Ω has about 5,000 indices. This allows only N_e=100 energy points per node. If only a small number of processors is available for a calculation requiring many energies, the internal memory allocation may prohibit the storage of all the energies. In this case, we have also constructed an outer loop over energy bins, i.e. the calculation is repeated and the Ω array written to the output file a number of times $N_t = N_e^{tot}/(N_p N_e^{max})$, where N_e^{max} is the maximum number of energy points permitted on a node and can be written as $N_e^{max} = 5 \times 10^5/(N_s(N_s+1)/2)$. This outer loop is also shown in Fig. 1 by the dotted lines.

III. TIMING CONSIDERATIONS

The time needed to complete a given scattering calculation for N_p processors, $T(N_p)$, can be subdivided into three separate timing categories as

$$T(N_p) = T_{rw}(N_p) + T_{mess}(N_p) + T_{comp}(N_p)\ . \tag{5}$$

The first term, $T_{rw}(N_p)$, is just the time required by node zero to read the input files and write the output files. Assuming that the outer-most loop over energy bins (see Fig. 1) is executed only once, this is independent of the number of nodes ($T_{rw}(N_p) = T_{rw} =$ constant).

The second term in Eq. 5 refers to the time used in broadcasting and receiving information between nodes, mainly the distribution of the input data from node zero to all other nodes. This can be written as

$$T_{mess}(N_p) = T_s + MT_w + T_h(h-1) \ , \tag{6}$$

where T_s is the start-up time, M is the size of the information to be sent in megawords, T_w is the per-word transfer time, h is the number of hops, and T_h is the hop time [6]. For complex cases, the M is roughly the sum of the sizes of the surface amplitudes, $w_{i\alpha}$, and the asymptotic coefficients, $C_{ij}^{(\lambda)}$. The dimension of the α index, N_b, is typically 20-50 times larger than the number of channels, N_c. And a calculation employing as many as $N_c = 100$ channels is not uncommon. The dimension of the λ index is typically about 10-20. For such a case, M is then about half a megaword.

The third term in Eq.5 is the actual time used to compute the collision strength. In order to minimize this time, the energy points must be distributed among the various nodes in an efficient manner. Otherwise, if the most time-consuming energy points are all assigned to one node, for instance, then the time required for the calculation is independent of the number of other processors available, since these will be idle while that one node computes. In this case, no speed-up will be observed as the number of nodes increases, and so $T_{comp}(N_p)$ is roughly constant.

The computational time required for a certain energy is strongly affected by the magnitude of the closed-channel energies; smaller values of $|E_i|$ take longer to compute. As one scans in energy through a threshold, such small values are encountered below each target threshold energy. We chose an interleaved partitioning of energies, as outlined in the previous section, in order that such time-consuming energy regions are equally partitioned among all the nodes, thereby ensuring an efficient allocation of energy points. We found for all the test cases to be discussed below, even those with a small number of energy points, that our scheme for partitioning energies was quite efficient, that is, the computational time used on each node was about the same. Given that the energy points are efficiently distributed, then, the total computational time taken is proportional to the number of energy points per node, $N_e = N_e^{tot}/N_p$. If the time required for the calculation using one node is T_c, then the total computational time is then

$$T_{comp}(N_p) = T_c \frac{N_e^{tot}}{N_p} \ , \tag{7}$$

so that the total time required for a given calculation, including reading, message passing, and computation, can be written as

$$T(N_p) = T_{rw} + T_{mess}(N_p) + T_c \frac{N_e^{tot}}{N_p} \ . \tag{8}$$

It is seen that, provided that $N_e^{tot} T_c >> T_{mess}$, linear scaling is achieved,

$$T(N_p) \sim T(1)/N_p \ . \tag{9}$$

Even for the case of a large numbers of channels, where T_{mess} is quite large, this scaling will be achieved as long as the collision strength is needed at a large number of energy points (when many narrow resonances must be resolved). But if the number of energies, N_e, is small, non-linear scaling with the number of nodes may be observed. Indeed, in the extreme case where there are only a couple of energy points per node, the time actually increases with the number of nodes.

IV. SIMPLE TEST CASE

The first case we tried on both the iPSC/860 and the Parallel Virtual Machine (PVM) software using a cluster of Sparc10 workstations was that of the $3s \rightarrow 3p$ electron-impact excitation of Ar^{7+}. The specifics of this calculation were described in an earlier study [7], but in the present case, only two target states were included ($N_s = 2$). Because of this, the computational time was minimal, as was the size of the messages passed (the $w_{i\alpha}$ and $C_{ij}^{(\lambda)}$ arrays).

We plot the elapsed time vs. number of processors in Fig. 2 for both machines at two different values for the number of energy points, N_e. The solid lines represent linear-scaling behavior extrapolated from the lowest possible number of nodes (one for the Sparc10 cluster, two for the iPSC/860). The Sparc10 cluster results for $N_e = 128$ show that nonlinear scaling is observed even in going from 1 to 2 nodes, and eventually no speed-up is observed by increasing the number of nodes. The iPSC/860 machine runs about 4 times slower per processor as evidenced by the factor of four difference in elapsed times between the two machines using two nodes. But the message-passing overhead is significantly less on the iPSC/860. This can be seen by noting that in Fig. 2 the divergence from linear scaling is not as severe for the iPSC/860 results.

The upper set of data points represent the case where $N_e = 1280$ energy points. For either system, the degree of nonlinear scaling has been substantially reduced. This behavior can also be inferred from Eq. 5. The size of the inter-node message sent does not increase in going to a larger number of energy points, and therefore $T_{mess}(N_p)$ remains constant for a given number of nodes. But the amount of computational time, $T_{comp}(N_p)$, is proportional to N_e. Therefore, $T_{comp}(N_p)$ comprises a greater fraction of the total time is governed by the linear scaling form of Eq. 9.

We compare in Table I the times required to run the case $N_e = 1280$ on various machines. While neither the Sparc10 or iPSC/860 speeds are comparable to a Cray-2, the use of many processors leads to a substantial savings in time.

V. LARGE TEST CASE

In order to look at the performance of our modified code for a typical, large case, we chose to study the $4s^2\ {}^1S_0 \rightarrow 4s4p\ {}^3P_{0,1,2}$ electron-impact excitation of Kr^{6+} . This case follows from recently completed work [8], where a large number of energy points ($N_e = 2176$) was required in order to resolve the resonance structure. Also, we used $N_s = 14$ target states, which led to as many as $N_c = 60$ channels and up to $N_b = 10^3$ basis functions. As mentioned in Section III, the computational time depends strongly on how close the energy is to a target threshold energy. Whereas for the Ar^{7+} case, all energies were above the only excited threshold, for this case we needed to scan through various thresholds. We found that the computational time required per energy was more than two orders of magnitude greater than for the Ar^{7+} case. The size of the messages to be passed was also greater for this case, but only by about one order of magnitude. Thus linear scaling should be achieved more readily than for the smaller case.

From the results shown in Fig. 3 using the Sparc10 cluster, we indeed observe this scaling for $N_e = 2176$ energy points, but also for as few as $N_e = 220$ points. These results only consider the use of up to 16 nodes with the Sparc10 cluster. Also shown in Fig. 3 are the times on the iPSC/860 using both 16 and 64 nodes, where even at this high number of nodes fairly good scaling is observed. Comparison of these times with the performance of a Cray-2 are listed in Table II. The Cray-2 results were run at Lawrence Livermore National Laboratory using a standard fortran compiler. Even though some automatic micro-tasking was performed, we did not invoke the multi-tasking compiler options. On the iPSC/860 machine using 64 nodes, this calculation was performed about three times as fast as on the Cray-2.

VI. CONCLUSION

We have shown that linear scaling of the total time with the number of processors can been achieved with our modified aymptotic code. The results for a large problem were thus obtained at a fraction of the real time required one processor. For this case, there were 14 atomic target states included. The largest cases we have run included 26-states for the electron-impact excitation of Sc^{2+} [9] and Ti^{3+} [10], which required about 3 hours of CPU time on a Cray-2 for $N_e = 500$ energy points. The new generation of parallel machines, such as Cray's T3D, Intel's Paragon, or Thinking Machines CM-5, have processing speeds comparable to a Cray-2, and more than 1 megaword of internal memory. If one were allocated say 1000 nodes for a 3-hour job, a 500-energy-point calculation could be performed including as many as 260 atomic target states, assuming that the time required scales as the number of target states cubed ($T \sim N_s^3$). Thus, many atomic scattering problems that had been considered too time-consuming for practical purposes may soon be investigated on these new machines.

We have described the parallelization of only one program belonging to the general R-matrix suite of codes [1,2], namely the STGF program. Our future goal is to modify the program STG3 [2], which diagonalizes the Hamiltonian matrix. This is the most time-demanding program for complex systems, so it's parallelization is essential in order to address more difficult atomic collision problems. For larger systems, the main obstacle will be storing the Hamiltonian in internal memory. The iPSC/860 allows only one megaword of internal memory per node, which means that even moderate calculations require the partition of the Hamiltonian onto several nodes. Future generations of parallel machines will hopefully increase this maximum allotment, as well as improve the processor speeds, in order to facilitate an efficient parallelization of codes with such memory requirements.

ACKNOWLEDGMENTS

We would like to thank the members of the Opacity Project for the use of their R-matrix programs. This work was supported in part by the U.S. Department of Energy, Office of Fusion Energy, under Contract No. DE-FG05-86-ER53217 with Auburn University.

REFERENCES

[1] P. G. Burke and K. A. Berrington, "Atomic and Molecular Processes: an R-matrix Approach", (IOP Publishing, 1993).

[2] K. A. Berrington, P. G. Burke, K. Butler, M. J. Seaton, P. J. Storey, K. T. Taylor, and Yu Yan, J. Phys. B**20**, 6379 (1987).

[3] M. J. Seaton, Rep. Prog. Phys. **46**, 167 (1983).

[4] D. P. O'Leary and G. W. Stewart, *Comm. of the ACM* **28**, 840 (1985).

[5] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan and Kauffman, p. 21 (1989).

[6] T. H. Dunnigan, *Communication Performance on the Intel Touchstone Delta Mesh*, Oak Ridge National Lab Report ORNL/TM-11983 (1992).

[7] N. R. Badnell, M. S. Pindzola, and D. C. Griffin, Phys. Rev. A **43**, 2250 (1991).

[8] T. W. Gorczyca, M. S. Pindzola, N. R. Badnell, and D. C. Griffin, Phys. Rev. A **51**, 488 (1995).

[9] M. S. Pindzola, T. W. Gorczyca, N. R. Badnell, D. C. Griffin, M. Stenke, G. Hofmann, B. Weissbecker, K. Tinschert, E. Salzborn, A. Müller, and G. H. Dunn, Phys. Rev. A **49**, 933 (1994).

[10] T. W. Gorczyca, M. S. Pindzola, D. C. Griffin, and N. R. Badnell, J. Phys. B **27**, 2399 (1994).

FIGURES

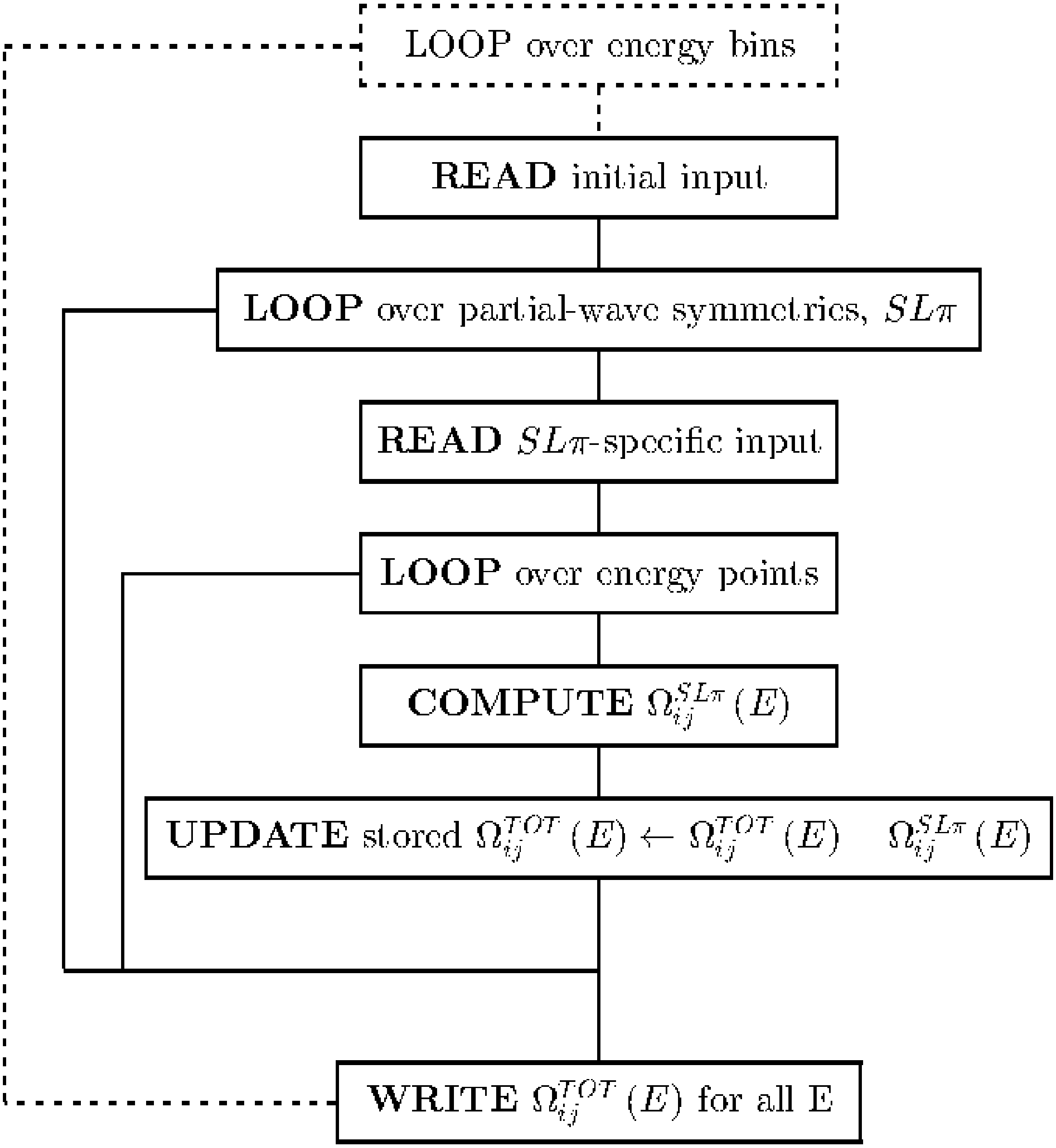

FIG. 1. Flow diagram for STGF program.

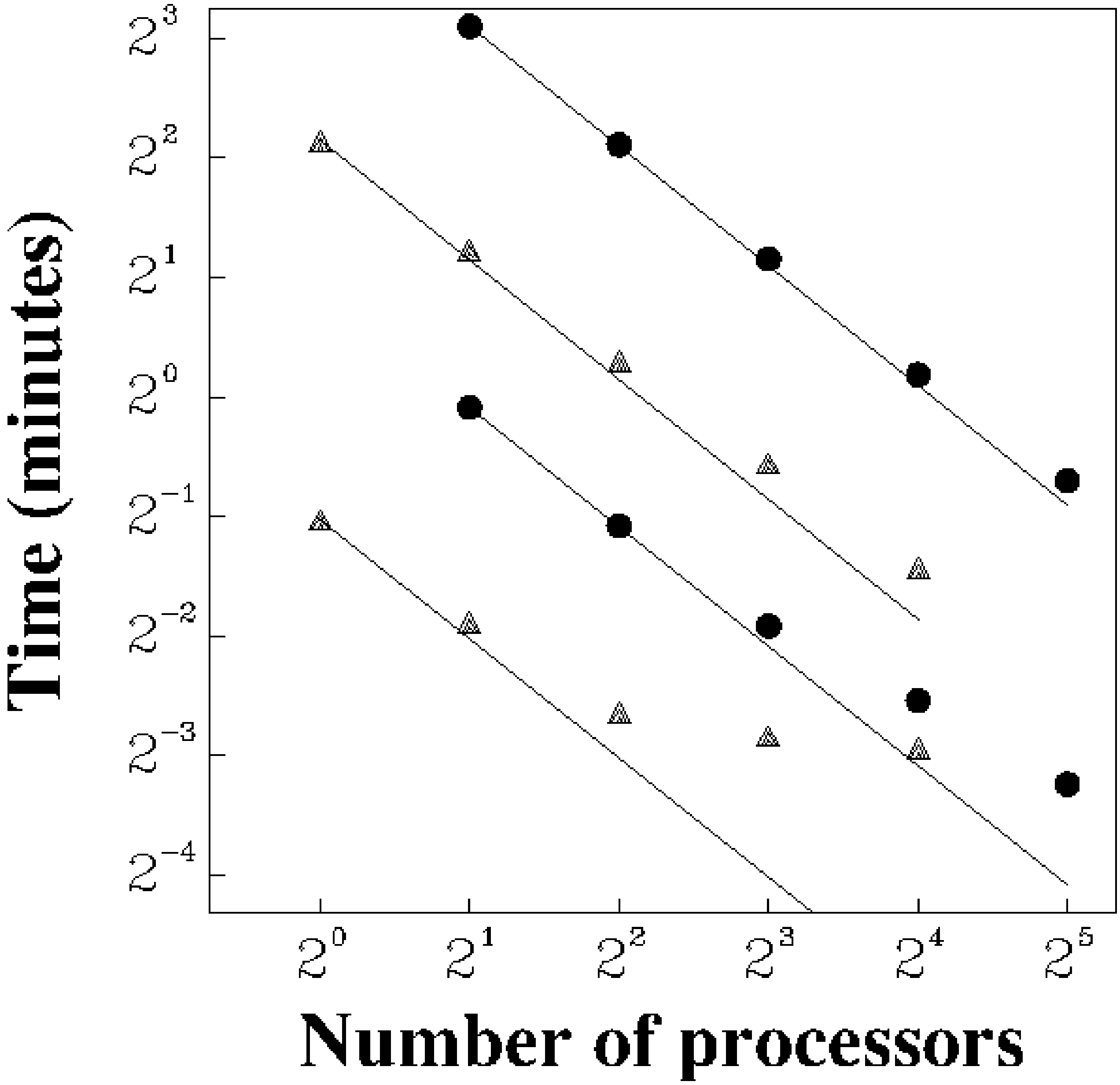

FIG. 2. Timing results for the Ar7+ test case versus number of processors. Circles: results using the iPSC/860; triangles: results using the Sparc10 cluster. Lower curve for each: N_e = 128 energy points; upper curve: N_e = 1280 energy points.

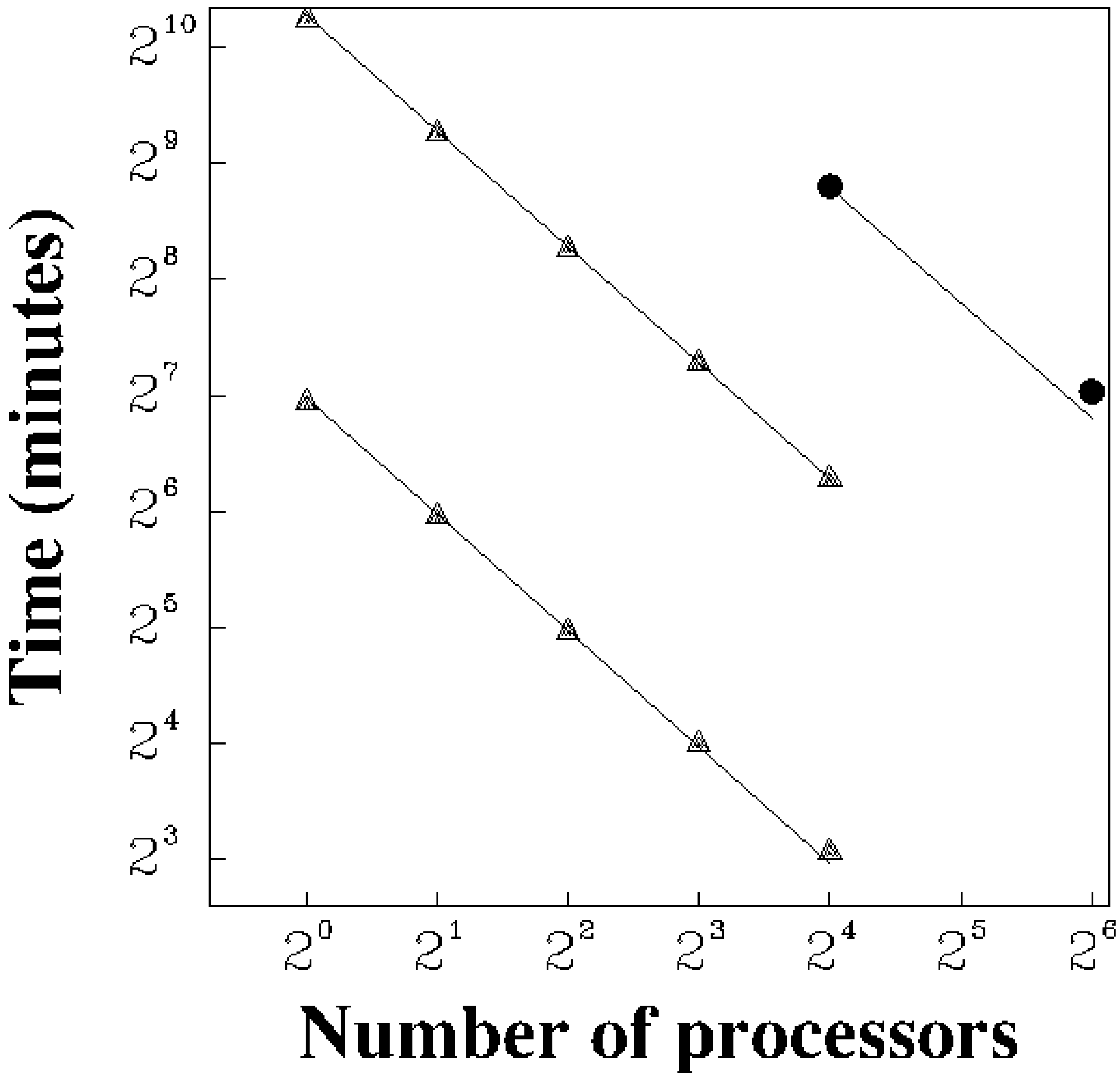

FIG. 3. Timing results for the Kr6+ test case versus number of processors. Triangles indicate results using the Spacr10 cluster for N_e = 220 (lower) and N_e = 2176 (upper) energy points. Circles are results using the iPSC/860 for N_e = 2176 energy points.

TABLES

TABLE I. Timing comparisons for Ar^{7+} test case ($N_e = 1280$ energy points)

Machine	Total time (minutes)
Cray-2	2.03
Sparc10	4.41
16 Sparc10's (PVM)	0.37
iPSC/860 (2 nodes)	8.57
iPSC/860 (32 nodes)	0.62

TABLE II. Timing comparisons for Kr^{6+} test case ($N_e = 2176$ energy points)

Machine	Total time (minutes)
Cray-2	376.8
Sparc10	1240.3
16 Sparc10's (PVM)	79.1
iPSC/860 (16 nodes)	446.6
iPSC/860 (64 nodes)	130.8