Theor Chim Acta (1993) 84: 315–323

Theoretica Chimica Acta

Harnessing the killer micros: Applications from LLNL's massively parallel computing initiative*

James Belak

Massively Parallel Computing Initiative, University of California, Lawrence Livermore National Laboratory, Livermore, CA 94550, USA

Received October 1, 1991/Accepted December 16, 1991

Abstract. Recent developments in microprocessor technology have led to performance on scalar applications exceeding traditional supercomputers. This suggests that coupling hundreds or even thousands of these "killer-micros" (all working on a single physical problem) may lead to performance on vector applications in excess of vector supercomputers. Also, future generation killer-micros are expected to have vector floating point units as well. The purpose of this paper is to present an overview of the parallel computing environment at Lawrence Livermore National Laboratory. However, the perspective is necessarily quite narrow and most of the examples are taken from the author's implementation of a large-scale molecular dynamics code on the BBN-TC2000 at LLNL. Parallelism is achieved through a geometric domain decomposition – each processor is assigned a distinct region of space and all atoms contained therein. As the atomic positions evolve, the processors must exchange ownership of specific atoms. This geometric domain decomposition proves to be quite general and we highlight its application to image processing and hydrodynamics simulations as well.

Key words: Killer micros – MPCI – Vector applications

1. Introduction

Figure 1 is taken from a talk Eugene D. Brooks III presented at Supercomputing '90 entitled "Attack of the Killer-Micros." The figure is a plot of relative performance (to a Cray-1[1]) of a single CPU versus the year the machine appeared. The plot for the conventional supercomputers[2] appears to level off

* Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48

[1] Cray-1 and Cray-YMP are trademarks of Cray Research Inc

[2] A conventional superconductor is what we normally think of as a mainframe dedicated to crunching numbers. This type of machine is typified by the current generation Cray-YMP supercomputer

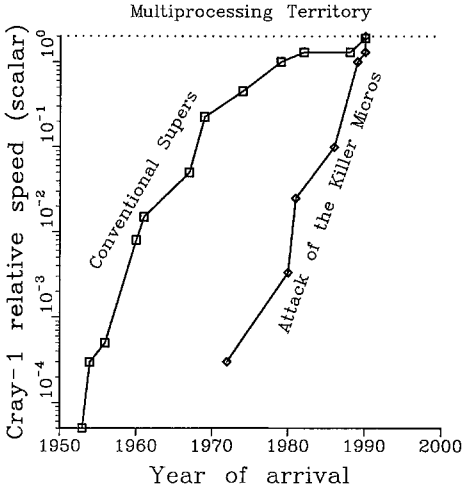

Fig. 1. A plot of the relative performance on a scalar application of a single cpu versus delivery date. The left hand curve is for conventional (mainframe) supercomputers, while the right hand side is for microprocessor based machines

with respect to the logarithmic vertical axis, while the performance of microprocessors continues to increase exponentially with no end in sight. Current generation microprocessors have increased 2–4 fold in performance during the time period since this figure was generated. These microprocessors are now at the performance limit set by the bandwidth to memory and we are observing hierarchical memory systems with fast (though still small) on chip cache, slower commodity local memory and slower-yet far away memory. This far away memory may belong to other processors with access through an interconnection network. Optimal programming of such machines involves migrating data to local memory before executing numerically intensive tasks. This performance limit due to memory bandwidth is being realized today with Intel i860[3] based machines – we can not seem to keep the hungry micro fed! However, we anticipate that microprocessor manufacturers will soon take a lesson from the manufacturers of traditional supercomputers and introduce interleaved (on chip) memory systems. This interleaved memory should provide at least one order of magnitude improvement in memory bandwidth and help quench the hunger of today's generation killer-micros. Tomorrow's generation may require new solutions.

2. LLNL's massively parallel computing initiative

The goal of the Massively Parallel Computing Initiative (MPCI) at LLNL is to provide a research and development environment to aid in the design of algorithms (and code) that are scalable to highly parallel machines – machines with at least O(100) processors. The issue of scalability is central to the optimal use of parallel computers. For example, an application that achieves 90% efficiency with 100 processors may only achieve 10–20% efficiency with 1000 processors (Amdahal's law). In order to achieve 90% efficiency with 1000 processors we must achieve 99% efficiency with 100 processors. The MPCI also

[3] i860 is a trademark of Intel Corp

provides a flexible programming environment with access to both shared memory and message passing programming models. Furthermore, we recognize that the optimal implementation of a specific application onto a parallel machine requires intimate knowledge of the underlying physics. For this reason, the MPCI effort is leveraged through the participation of a large number of LLNL's research staff with only a small core staff required to support the programming models and the day-to-day maintenance of the machine.[4]

Our current development machine is the BBN TC2000.[5] The MPCI TC2000 consists of 126 (plus two hot spares) fast RISC microprocessors (Motorola 88100). Each processor is located on a separate board, along with 16 megabytes of local memory. The nodes are interconnected by a scalable "butterfly" switch. At boot time, some of each processor's local memory (6 megabytes) is allocated to an interleaved shared memory pool. It takes about four times longer to access this shared memory through the switch than to access private memory, local to the processor's board. Thus, an efficient computer code must use private memory for its computationally intensive tasks.

The development system on the MPCI TC2000 is aimed at a multi-user and multi-tasking environment. A small number of nodes (8) are dedicated to a public cluster. These run the familiar UNIX operating system and perform the editing, compiling and job control that defines the user's interface to the machine. The remaining nodes (118) are assigned to a gang scheduled cluster. This is the cluster where parallel programs are executed. The gang schedular assures that each user's task is run is a timely and fair-share manner. The parallel programming tools on the machine consists of BBN's extensions to FORTRAN, the Parallel C Preprocessor (PCP) [1, 2] and its extension to FORTRAN (PFP), an implementation of message passing (LMPS) based upon the Argonne message passing system [3], and various utilities to monitor an executing program. We have chosen the C programming language for our variable particle molecular dynamics primarily because, as yet, the FORTRAN programming language does not support the constructs required for an efficient implementation. In this report, we explore the utility of interleaved shared memory and the PCP paradigm for the implementation of molecular dynamics algorithms. Message passing schemes for molecular dynamics are concurrently being explored on the TC2000 at LLNL by Tony DeGroot.

PCP provides an extension of the single-program-multiple-data (SPMD) programming model in the familiar C programming language. Each processor executes the same code and flow control is placed into the hands of the programmer. PCP introduces the concept of a "team" of processors. A team may split into sub-teams in order to divide up work. Each team has one master processor. We find the master block (a section that only the master enters) most useful in performing serial work on shared memory – work that all processors must know about before the calculation can proceed. Flow synchronization is obtained through the **barrier** statement. Every processor reaching a **barrier** waits until all members of its team (including the master) reach that **barrier**. A fast waiting algorithm has been implemented for PCP runtime support. Additional

---

[4] For an overview of all applications being developed through LLNL's MPCI effort, the reader is referred to the MPCI annual report (UCRL-ID-107022). A copy may be obtained by writing to: Chris Malone, MPCI L-416, Lawrence Livermore National Laboratory, P.O. Box 808, Livermore CA 94550

[5] TC2000 is a trademark of Bolt Beranek and Newman Inc

flow control for critical sections is accomplished with locks. A critical section is a region of the code in which many processors access the same resource and, to allow them to do so, would corrupt the results. Accumulating partial sums into a shared sum is a commonly encountered example. PCP provides the **lock(&lock_variable)** and **unlock(&lock_variable)** functions to isolate critical sections. The lock variable is stored in shared memory. The first processor entering the critical section sets the lock variable to **locked** and proceeds with the calculation. Meanwhile, the remaining processors test the lock variable to see whether it is locked. When the first processor finishes the calculation, it sets the lock variable to **unlocked**. The next processor to find it unlocked immediately locks it and proceeds with the calculation.

Parallelism is accomplished with the **forall** loop. The **for** loop in C is similar to the **do** loop in FORTRAN. The **forall** loop divides the indices of a **for** loop evenly amongst the available processors. Each processor does the work for the value of the index it knows about. Possibly one of the most useful aspects of PCP is the transparent access to both shared and private memory. Declarations are made with the **private** and **shared** storage class modifiers and dynamical memory allocation is made using the **prmalloc** and **shmalloc** functions.

3. Molecular dynamics modeling

Molecular dynamics (MD) modeling is very simple in principle [4]. Given the positions of all of the atoms, calculate the force on each atom due to its neighbors and advance the positions with a finite difference integration scheme. Both predictor-corrector and central difference are commonly used. In our simulations, we employ an embedded atom method (EAM) [5] to express the forces between the atoms in a simple metal. The total potential energy is written as:

$$\Phi_{\text{total}} = \tfrac{1}{2} \sum_{i,j} \phi(r_{ij}) + \sum_i F(\varrho_i) \tag{1}$$

with

$$\varrho_i = \sum_{j \neq i} f(r_{ij}). \tag{2}$$

The first term is the usual two body interaction energy and the second term is the energy required to embed the atoms into the local electronic charge density (ϱ_i). The Newtonian equations of motion for the embedded atom method are:

$$m \frac{d^2 x_k}{dt^2} = - \sum_{j \neq k} (\phi'(r_{kj}) + (F'(\varrho_k) + F'(\varrho_j)) f'(r_{kj})) \frac{x_k - x_j}{r_{kj}}. \tag{3}$$

These equations are inherently nonlocal – they depend on both the embedding density ϱ_k and ϱ_j. They must be solved in a two step manner. The embedding density at all the atomic sites is evaluated first, then the forces may be evaluated. The equations are integrated by approximating the time derivative by a central difference:

$$\frac{d^2 x}{dt^2} \approx \frac{x(t + \Delta t) - 2x(t) + x(t - \Delta t)}{\Delta t^2} \tag{4}$$

with a time-step (Δt) of about 1/25 of the vibrational period (τ_E).

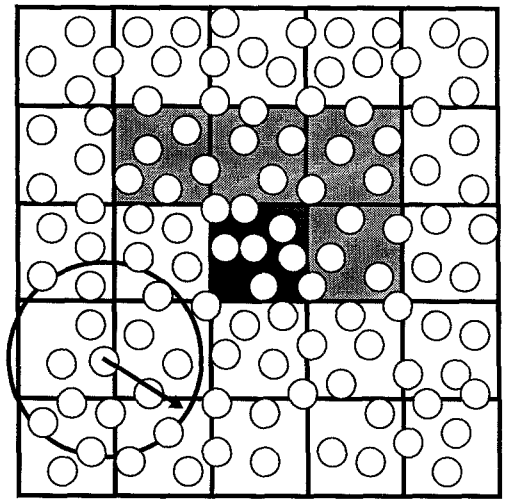

Fig. 2. A typical molecular dynamics simulation cell. The atom in the lower left hand corner has a circle surrounding it indicating the cut-off range used in short range potential models. The large simulation cell is subdivided into many small sub-cells, the size of which is determined by the interaction range. All atoms in the dark shaded (center) sub-cell interact with all atoms in the lighter shaded sub-cells

The vast majority of computer time spent executing any molecular dynamics program is spent calculating forces. In order to understand parallel MD strategies, it is useful to review how forces are calculated. In this work, we consider only short ranged forces. The important class of long ranged forces (Coulomb forces) requires each atom to interact with every other atom and the implementation onto parallel computers involves systollic loops [6]. Two methods have been developed to treat short ranged forces efficiently. In the first method, a list of neighbors within a cut-off distance (see Fig. 2) is maintained for every atom within the MD cell. However, the memory requirements of the method is prohibitively expensive for large system sizes. An alternative method is to subdivide the large computational cell into many small sub-cells. The size of these sub-cells is determined by the interaction cut-off [7]. Atoms within each sub-cell interact with atoms from neighboring sub-cells only (as shown in Fig. 2). In our initial implementation [8], we placed the entire problem into shared memory and parallelized over these sub-cells. A processor copies into its local memory the positions of all atoms from one sub-cell and its neighbors. It then proceeds to calculate forces. An advantage of this method is that the amount of available parallelism is always much greater than the number of processors. However, the efficiency is severely effected by the large amount of communication taking place. In our current implementation, we assign a domain (a connected set of sub-cells) to each processor. That processor is responsible for all atoms within its domain and stores these positions in its local memory. At the start of a time step, the processor gathers from shared memory all information for sub-cells immediately outside its domain (see Fig. 3). This information is stored in local "phantom" sub-cells to be used in the force calculation. At the end of the time step, the processor stores into shared memory all information for sub-cells immediately inside the boundary of its domain. This is the information that neighboring processors require for the next time step. The same algorithm may be implemented within a message passing programming model. However, care must be taken with the corner sub-cells. These require two hops in the message passing for two dimensional simulations and three hops for three

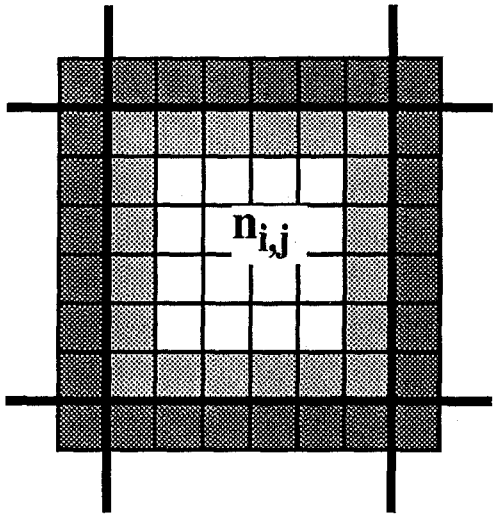

Fig. 3. This figure illustrates the geometric domain decomposition of the large MD cell. A domain is a connected set of sub-cells. The darker shaded sub-cells belong to neighboring processors and represent information that must be gathered at the start of a time step. The lighter shaded sub-cells (inner boundary) represent information that must be communicated to neighboring processors at the end of each time step

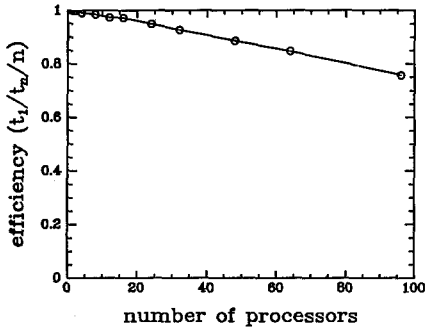

Fig. 4. The parallel efficiency of our molecular dynamics algorithm running on 1–100 processors and a fixed system size of 32768 atoms. The linear fall off with processor count is characteristic of contention for a shared resource (e.g. an inter-communication network)

dimensional simulations. The parallel efficiency of this algorithm for a fixed system size ($N = 32768$) is shown in Fig. 4. The figure shows the generic form:

$$e = \frac{1}{1 + \frac{t_{comm}}{t_{calc}}} \approx 1 - \frac{t_{comm}}{t_{calc}} \tag{5}$$

with $t_{calc} \approx t_0/p$ ($p = \#$ of processors)

$$e \approx 1 - p\frac{t_{comm}}{t_0}. \tag{6}$$

The resulting efficiency (80% at 100 processors) may be improved by increasing the system size (availabe parallelism). However, we suspect that there are still important details concerning communication time (t_{comm}) that we need to finely tune.

4. Image processing and hydrodynamics modeling

The implementation of image processing [9] and hydrodynamics simulations [10] is via a similar domain decomposition scheme as has been outlined for molecular

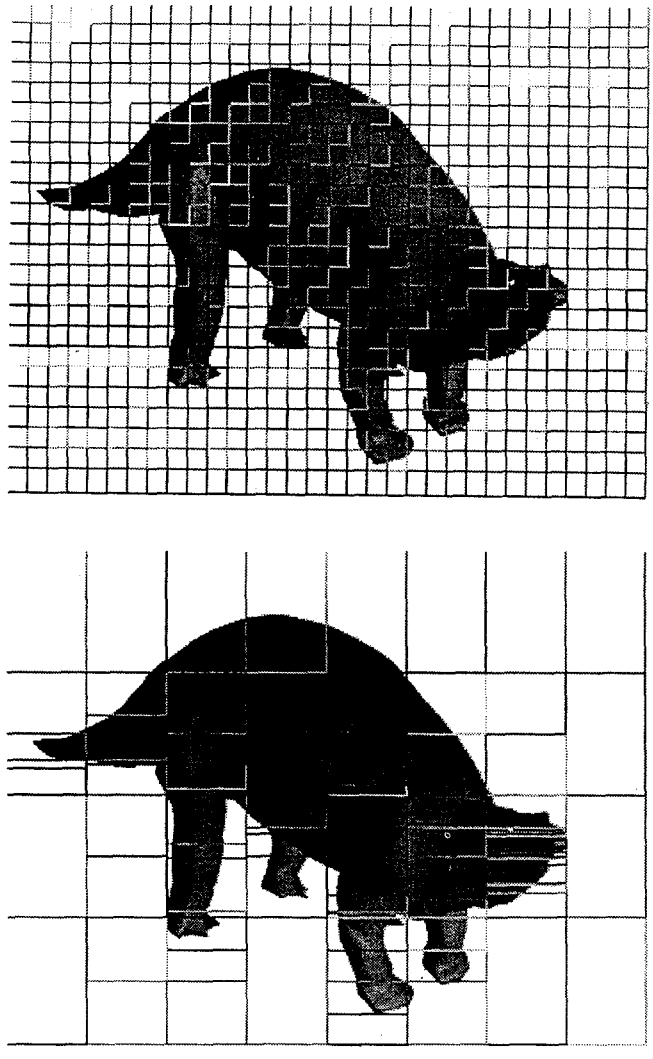

Fig. 5. Two geometric domain decomposition schemes used in image processing. The *upper* figure demonstrates a fixed decomposition scheme while the *lower* figure demonstrates an adaptive decomposition scheme

dynamics. During image rendering, the problem is first transformed into "image space," a coordinate system with one axis along the direction from the viewpoint (eye) to the object. The domain decomposition takes place in this image space. Each processor is assigned a collection of pixels and performs the rendering for all pixels within its domain (see top of Fig. 5). Unfortunately, fast algorithms that exploit nearest neighbor pixel information must be abandoned. Furthermore, each processor must have access to the entire object space. For most images, there are regions that require significantly more computational effort than others. In such cases, the efficiency may be improved by introducing an adaptive decomposition. This is illustrated at the bottom of Fig. 5. The initial image space is decomposed into p domains. When a processor finishes all work for its domain, it queries the system for the domain with the most work remaining (# of pixels not rendered). It then divides that domain with the processor working on it. The process continues until the image is rendered.

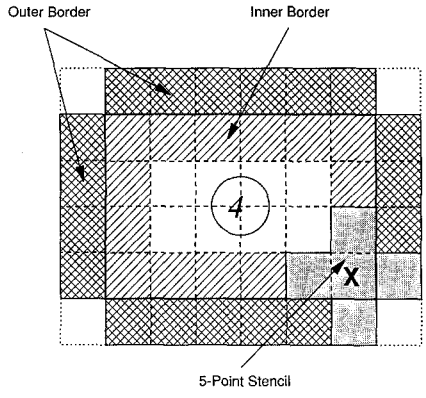

Fig. 6. A geometric domain decomposition stencil commonly used in Eulerian hydrodynamics models. The 5-point stencil does not require corner data to be communicated across processors. Because the calculational work per node is small, care must be taken to minimize the interdomain communication

Hydrodynamics simulations suffer a drawback not encountered in MD simulations. Unlike MD, the amount of work per node is typically small. Usually just the differencing on a fixed grid (see Fig. 6). Domain decomposition still proves useful for large scale applications such as global climate modeling. Because of the small calculation time, extra care must be taken to minimize interdomain communication.

5. Conclusions

Geometric domain decomposition schemes provide a common thread between the implementation of several distinct applications onto parallel computers. As our available parallel machines become larger (more nodes), we expect to study larger problems with high efficiency by maintaining a constant domain size per processor. For molecular dynamics simulations, this means larger length scales but NOT longer time scales. The time step is limited by the clock cycle of the processor and we should never expect to obtain more than one time step per clock cycle. Hydrodynamics provides a two-fold problem. Typically, the size of the problem is fixed (the entire earth in the global climate modeling) and the additional parallelism is used to improve resolution so that the number of finite element nodes per domain remains fixed. Unfortunately, the "size" of the node determines the time step via the Courant condition – small nodes demand smaller time steps. Thus, the computational work (number of time steps) required to simulate a fixed period of time (say 100 years) will always increase! There are additional problems that we have not discussed in detail. For example, how long does it take to load one of these large problems into memory? It seems silly to spend several hours loading an application that might run for a few minutes!

Acknowledgements. It is a pleasure to acknowledge Eugene D. Brooks III and all of the MPCI core staff for providing an efficient parallel computing environment to work within.

References

1. Brooks III ED (1988) PCP: A parallel extension of C that is 99% fat free. Lawrence Livermore Natl Lab Report UCRL-99673

2. Gorda B, Warren K, Brooks III ED (1990) Programming in PCP. Lawrence Livermore Natl Lab Report UCRL-MA-107029
3. Welcome T (1990) Programming in LMPS. Lawrence Livermore Natl Lab Report UCRL-MA-107031
4. For advanced details of molecular dynamics modeling see: Heerman DW (1989) Computer simulation methods in theoretical physics, Second edn (Springer-Verlag, Berlin); Allen MP, Tildesley DJ (1987) Computer simulation of liquids. Oxford Univ Press, Oxford
5. Daw MS, Baskes MI (1984) Embedded atom method: Derivation and application to impurities, surfaces, and other defects in metals, Phys Rev B 29:6443
6. Fincham D (1987) Parallel computers and molecular simulation, Molecular Simulation 1:1
7. Rapaport DC (1988) Large-scale molecular dynamics simulation using vector and parallel computers. Computer Phys Reports 9:1
8. Belak J (1991) A parallel implementation of a molecular dynamics algorithm using the PCP programming paradigm and its application to orthogonal metal cutting, in: The 1991 MPCI Yearly Report, UCRL-ID-107022, Lawrence Livermore Natl Lab
9. Whitman S, Sadayappan P (1991) Computer graphics rendering on a shared memory multiprocessor, in: The 1991 MPCI Yearly Report, UCRL-ID-107022, Lawrence Livermore Natl Lab
10. Procassine RJ, Dannevik WP (1991) A shared-memory implementation of a global ocean model on a MIMD parallel computer, in: The 1991 MPCI Yearly Report, UCRL-ID-107022, Lawrence Livermore Natl Lab