Shared-Memory Implementation of an Irregular Particle Simulation Method

THOMAS RAUBER  GUDULA RÜNGER *  CARSTEN SCHOLTES

Computer Science Dep., Universität des Saarlandes, 66041 Saarbrücken, Germany

Abstract. We investigate a parallel implementation of an irregular particle simulation algorithm. We concentrate on the issue which programming and system support is needed to yield an efficient implementation for a large number of processors. As execution platform we use the SB-PRAM, a shared memory machine with up to 4096 processors.

1 Introduction

Most investigations of irregular applications concentrate on implementations on shared-memory machines with a small or medium number of processors (usually not more than 64) [5, 6]. In this article, we consider the question whether a typical irregular applications can efficiently be executed on larger machines. As example application, we consider the particle simulator MP3D [7]. We investigate which issues are important for large speedup values, which system support should be available, and which programming effort must be invested. For hundreds or thousands of processors, it is essential to investigate the exploitable degree of parallelism which represents an upper bound for the achievable speedup. Moreover, sequential portions of the parallel implementations, the granularity of the computations, and load balancing issues have a much larger influence on the efficiency than for a smaller number of processors.

The investigations are executed on a simulator of the SB-PRAM. The SB-PRAM is designed for 128 physical processors which provide a total number of 4096 virtual processors seen by the programmer [1]. The machine provides a global shared memory with uniform access time, i.e., from a virtual processor's point of view, an access to the global memory takes the same time as two arithmetic operations, independently from the memory location that is addressed. Because of this memory organization, locality properties can be neglected and the investigations can concentrate on the exploitation of the maximum degree of parallelism and the usage of efficient parallel data structures. Thus, we can obtain an upper bound on the attainable speedup, which also takes into consideration the overhead for shared data structures and the task management overhead. The avoidance of sequentializations and a good load balance are supported by the powerful multiprefix operation provided by the SB-PRAM[3]

2 The Particle Simulation MP3D

MP3D is a particle simulator for fluid dynamics problems, which uses a uniform subdivision into cubic cells [2, 7, 5]. MP3D is used to study rarefied fluid flow

* author supported by DFG
around objects as they pass through the upper atmosphere at hypersonic speed. The significance of the method is the reduction of the interaction phase from $O(n^2)$ to $O(n)$. The program simulates a set of particles by computing their position and velocity in each time step. The space vehicle is represented as a flat object in a rectangular wind tunnel with openings at each end and reflecting walls on the remaining sides. Particles generally flow through the tunnel in positive $x$-direction. A particle may collide with the boundaries of the wind tunnel, the object, and other particles. Particles can collide with each other only if they reside in the same space cell. The program starts the simulation with a fixed number of particles that are moving with a constant wind velocity in $x$-direction. The density of the particles represents the resulting air pressure. Particles exiting the wind tunnel are replaced by new particles from a small reservoir, which enter the tunnel at random positions near the entrance.

The main data structures of the MP3D program are two large arrays. One array is used to store the particle information (position, velocity). A second array is used to store the cell information (collision probability, collision information, number and velocity statistics for particles entering the cell). Each timestep consists of five phases. The initialization phase resets the collision counters and the population counters of the space cells and recomputes the collision probabilities. The move phase moves all particles to new positions and computes collisions. In the add phase, new particles are entered into the tunnel to simulate a constant current of air. Particles in the reservoir are moved in the move reservoir phase. In the collide reservoir phase, the reservoir particles are paired and a collision is computed for each of the particle pairs. Tests with a sequential version of the program show that more than 96% of the total runtime is used for the move phase. This percentage is increasing for increasing numbers of processors.

The move phase computes a new position and velocity for each particle $\rho$ in the wind tunnel. Moving a particle may lead to a collision with a surface of an object or a boundary of the tunnel or with another particle. The program treats the different kinds of collisions one after another, i.e., for each particle, first a collision with a boundary is tested and then a collision with another particle. A collision with a surface may happen, if the cell $cell(\rho)$ in which particle $\rho$ is positioned contains a surface. If the particle hits the surface, its position and velocity are changed according to the laws of elastic impact. The collision between particles is computed by using cell tags of particles located in each cell and formulas expressing the collision probability. A particle $\rho$ sets the cell tag of $cell(\rho)$ if the tag information is empty. If $cell(\rho)$ already contains a tag for another particle $\rho'$, a collision between $\rho$ and $\rho'$ may happen. The collision in $cell(\rho)$ is computed according to a collision probability. If a collision takes place, the velocity of $\rho$ and $\rho'$ are changed accordingly and the tag of $cell(\rho)$ is cleared.

3 Implementation for small numbers of processors

The original implementation of MP3D from the SPLASH-1 benchmark suite was developed for a small or medium number $p$ of processors ($p \leq 64$) [5]. This is reflected in the implementation of the different phases. In the initialization phase, the 60 rows of inner cells, each one containing 14 cells, are statically distributed round-robin among all processors. The relatively large grain size of 14 cells results in a load imbalance if $60/p \notin N$. In the move phase, the set of particles to be simulated is split up into packets of 64 particles each, mainly
to improve the memory alignment and cache behavior on some architectures. The packets are distributed statically round-robin among all processors. Each processor simulates the particles of the packets assigned to it in turn. Since the particles assigned to a specific processor $q$ move around unpredictably in the wind tunnel, $q$ may have to access different cells' data during the simulation of different particles and even during the simulation of one particle. Basically, the simulation of one particle $p$ comprises the following steps: First, some particle related statistics of cell($\rho$) are updated. The access to the statistical data of cell($\rho$) is protected by a lock to avoid simultaneous updates by other processors. After this, the particle is moved, possibly into another cell. Second, the cell in which the particle now resides must be checked, whether it contains a boundary or not. If cell($\rho$) contains a boundary, an interaction with it is simulated making an additional lock necessary. The computation of the interaction may place $\rho$ into a third cell. This possibly new cell($\rho$) is then locked for a possible particle-particle collision; it must be checked whether it contains a collision particle $\rho'$ or not. If $\rho'$ exists, a collision between $\rho$ and $\rho'$ is computed. The lock of cell($\rho$) is released after the computation of the collision. The simulation of a particle-particle collision only affects the velocities of the particles $\rho$ and $\rho'$, not their positions, i.e., no further space cell can be entered. Thus, up to three different space cells may be accessed for the simulation of one particle in one simulation step and between two or three locks are used. The time to simulate a particle may differ considerably depending on whether collisions have to be computed. Thus, the statical assignment of particles to processors may lead to large differences in the work load, especially if each processor has to simulate only a few particles. Additionally, the distribution in packets of 64 particles may lead to an additional load imbalance if $\lfloor n/64 \rfloor / p \notin \mathbb{N}$. In the remaining three phases the particles to be added, the reservoir particles to be moved, and the pairs of reservoir particles to be collided, respectively, are distributed statically in a round-robin way among all processors.

Figure 1 (left) shows the resulting speedup values for different shared-memory multiprocessor platforms, the cache based KSR-1 and Stanford DASH, a bus-based Silicon Graphics 4D/380S (SGI), and the SB-PRAM. The values for the KSR-1 are taken from [7] and show a simulation run with 50000 particles, the values for the SGI are taken from [2] and show a simulation run with 8000 particles, the values for the DASH are taken from [4]. Both the DASH and the SB-PRAM runs show a simulation with 40000 particles. For the KSR-1, the speedup is decreasing for the KSR-1 with increasing numbers of processors, mainly because of contention for accesses to the shared space array. For the DASH, the speedups are poor because of frequent sharing and updates of space cells. As the number of processors increases, it becomes more and more likely that the newest version of a space cell being referenced resides in the cache of another processor. For the same reason, the speedup for the SGI is not increasing with increasing numbers of processors. The efficiency of the original implementation on the SB-PRAM is quite large because of the efficient realization of the locking mechanism with the help of the multiprefix operations. As shown in [3], the execution of a lock or unlock operation is independent from the number of participating processors, as long as no sequentializations occur. Moreover, the lack of spatial locality does not have a negative influence as it is the case for the KSR-1 or the DASH. For larger number of processors, the efficiency on the SB-PRAM is slightly decreasing because the probability for sequentializations through concurrent accesses to the same space cell by different processors is increasing. The efficiency on
the SB-PRAM is not optimal because of the overhead of the locking mechanism and because of the load imbalance caused by the static scheduling. Figure 1 (right) shows the runtime of restructured versions of the MP3D program. The versions on the KSR-1, SGI, and DASH result from major changes in the program to enhance the locality of MP3D, including a spatial decomposition among the processors, see [7, 2, 4] for a detailed description. The changes for the SB-PRAM are restricted to the use of efficient parallel data structures with dynamic scheduling and the use of multiprefix operations.

4 Implementation for large number of processors

Although the original implementation yields a good performance on the SB-PRAM, it can be considerably increased by reorganizations that increase the degree of parallelism and by using dynamic scheduling. The degree of parallelism is increased by using a fine grain size for the different tasks: In the initialization phase, the degree of parallelism is increased to 840 by assigning single cells to processors. Because the computational effort per space cell is the same, a static assignment of cells to processors can still be used. In the move phase, the degree of parallelism is increased to the number of particles to be simulated by removing the building of packets of 64 particles.

As discussed in the previous subsection, the static assignment of particles to processors may lead to a load imbalance. This can be removed by a dynamic assignment of particles to processors which can be realized by the use of a parallel loop. The loop is controlled by a shared counter $c$ which is accessed by the processors with a multiprefix operation to get the next particle to be simulated. Because of the use of the multiprefix operation, no sequentializations occur when different processors access the shared counter concurrently. The same effect can be obtained by the use of a central parallel task queue which can be accessed by all processors in parallel without sequentializations, see [3]. A further improvement is obtained when replacing locks to shared data structures by multiprefix operations if this results in the same behavior. This can be applied for the update of the cell statistics at the beginning of the move phase, because each processor contributes only informations of the particle that is currently simulated and because the statistical data is not used before the next simulation round. Locks
can also be replaced by multiprefix operations when particles are added near the entrance to get a constant air pressure and when particles are moved to the entrance if they leave the tunnel through the exit.

Figure 2 shows the percentage of the improvements on the MP3D program for runs with 8000 and 40000 particles, respectively. The figure shows the effects of the single changes compared to the original implementation. The following changes are investigated: `mpcells` shows the effect of increasing the degree of parallelism in the initialization phase by assigning single cells to the processors; a dynamic scheduling is applied by using a parallel loop; `align` shows the effect of increasing the degree of parallelism in the move phase by assigning single particles to the processors instead of packets of 64 particles; a static scheduling is used for the distribution; `mpdist` shows the effect of using the `align` variant with a dynamic scheduling which is obtained by a parallel loop; `mpsmpl` uses a multiprefix operation instead of a lock for the update of the cell statistics at the beginning of the move phase of each particle; `mpcoll` uses a multiprefix operation instead of a lock for the statistics at the end of each move phase.

The use of the `mpcells` option leads to a smaller runtime because of a better load balance. The improvement in percentage is increasing for increasing number of processors until the maximum degree of parallelism (840) is reached. The use of the `align` option leads to the largest reduction in the runtime because of a better load balance. The effect is larger than the effect of the `mpcells` option because the move phase contributes a much larger fraction to the global runtime than the initialization phase. The improvement in percentage is large if the load balance of the original implementation can be improved by a better distribution of the particles among the processors. This is, for example, the case for 8000 particles and 256 processors. In the original implementation, 125 processors get 64 particles each, the rest of the processors is idle. With the `align` option, each processor simulates 31 or 32 particles, respectively. The additional use of the `mpdist` option can lead to an additional improvement because the load is balanced even better by the dynamic scheduling. This effect has a considerable impact mainly in the case when each processor has to simulate only a few particles. If the number of particles per processor is large, the advantage of the dynamic scheduling diminishes, because the static scheduling already establishes a quite even distribution among the processors. For large numbers of particles per processor, the overhead of the dynamic scheduling even leads to a small increase of the runtime compared to the `align` variant. The `mpsmpl` option leads to a slight improvement because the direct use of a multiprefix operation
is faster than the use of a lock operation. The improvement is not increasing for increasing numbers of processors because the execution time of a lock or unlock operation is constant, independent from the number of participating processors. Moreover for up to 256 processors, there are usually only a few sequentializations through concurrent accesses to the same space cell when using the locking variant. For larger numbers of processors ($p > 256$), the percentage of improvement of the \texttt{mpspmpl} option is getting larger because the possibility for sequentializations increases (not shown in the figure). The \texttt{mpcool} option has only a small influence on the runtime because the update of the statistics at the end of each move phase only contributes a small fraction to the global runtime. The use of a multiprefix operation instead of a lock operation for the update of the number of particles in the wind tunnel and the number of particles in the reservoir at the end of each simulation round does not have a significant influence on the runtime and, therefore, is not shown in the figure.

5 Conclusions

For most irregular applications, it is difficult to achieve good speedup values for large number of processors. One of the main reasons lies in the lack of locality of these applications. The negative influence of missing locality is usually getting worse for increasing number of processors. To enhance the spatial locality and the resulting speedup values, major reorganizations are required for most applications. We use the MP3D program as example and show that much better results can be obtained if the underlying hardware provides appropriate operations to implement shared data structures and to support dynamic task scheduling. Another advantage of this approach is that the resulting programs are much easier to write and to maintain.

References


This article was processed using the \texttt{E\textsc{T}X} macro package with LLNCS style