Multi-core Strategies For Particle Methods

John R. Williams, David Holmes and Peter Tilke*

Massachusetts Institute of Technology, *Schlumberger-Doll Research Laboratory

This paper discusses the implementation of particle based numerical methods on multi-core machines. In contrast to cluster computing, where memory is distributed across machines, multi-core machine can share memory across all cores. Here general strategies are developed for spatial management of particles and sub-domains that optimize computation on shared memory machines. In particular, we extend cell hashing so that cells bundle particles into orthogonal tasks that can be safely distributed across cores avoiding the use of “memory locks” while still protecting against race conditions. Adjusting task size provides for optimal load balancing and maximizing cache hits. Additionally, the way in which tasks are mapped to execution threads has a significant influence on the memory footprint and it is shown that minimizing memory usage is one of the most important factors in achieving execution speed and performance on multi-core. A novel algorithm called H-Dispatch is used to pipeline tasks to processing cores. The performance is demonstrated in speed-up and efficiency tests on a smooth particle hydrodynamics (SPH) flow simulator. An efficiency of over 90% is achieved on a 24-core machine.

INTRODUCTION

Recent trends in computational physics suggest a rapidly growing interest in particle based numerical techniques. While particle based methods allow robust handling of mass advection, this comes at the computational cost of managing the spatial interactions of particles. For problems involving millions of particles, it is necessary to develop efficient strategies for parallel implementation. A variety of authors have reported parallel particle codes on clusters of machines (see for example Nelson et al [1], Morris et al [2], Sbalzarini et al [3], Walther and Sbalzarini [4], Ferrari et al [5]). The key difference between parallel computing on clusters and on multi-core is that memory can be shared across all cores on today’s multi-core machines. However, there are well known challenges of thread safety and limited memory bandwidth. A variety of approaches to programming on multi-core have been proposed to-date. Concurrency tools from traditional cluster
computing, like MPI [6] and OpenMP [7] have been used to achieve fast take-up of the new technology. If one runs an MPI process on each core, the operating system guarantees memory isolation across processes. Communication of information across process boundaries requires MPI messages to be sent, which is relatively slow compared with direct memory sharing.

In response to this, Chrysanthakopoulos and co-workers [7, 8] have implemented multi-core concurrency libraries using Port based abstractions. These mimic the functionality of a message passing library like MPI, but use shared memory as the medium for data exchange, rather than exchanging serialized packets over TCP/IP as is the case for MPI. Such an approach provides extensibility in program structure, while still capitalizing on the speed advantages of shared memory. In an earlier paper, Holmes et al. [9] have shown that a programming model developed using such port based techniques provides significant performance advantages over MPI and OpenMP. In this paper, we apply the programming model proposed in [9] to the parallelization of particle based numerical methods on multi-core.

PARTICLE BASED NUMERICAL METHODS

Mesh based numerical methods have been the cornerstone of computational physics for decades. Here, integration points are positioned according to some topological connectivity or mesh to ensure compatibility of the numerical interpolation. Examples of Eulerian mesh based methods include finite difference (FD) and the lattice Boltzmann method (LBM), while Lagrangian examples include the finite element method (FEM). While powerful for a wide range of problems, mesh limitations for problems involving large deformation and complex material interfaces has led to significant developments in meshless and particle based methodologies. For such methods, integration points are positioned freely in space, capable of advecting with material in a Lagrangian sense. For methods like molecular dynamics (MD) and the discrete element method (DEM), such points represent literal particles (atoms and molecules for MD and discrete grains for DEM), while for methods like dissipative particle dynamics (DPD) [10], smooth particle hydrodynamics (SPH) [11], wavelets [12], and the reproducing kernel particle method (RKPM) [13], the particle analogy is largely figurative. For such methods, parti-
cles provide positions at which to enforce a partition of unity. By partitioning unity across the particles, continuity can be imposed without a defined mesh, allowing such methods to represent a continuum in a generalized way. In Eulerian mesh based methods, such as FD and LBM, continuity is inherently provided by the static mesh, while for Lagrangian mesh based approaches like FEM, continuity is enforced through the use of element shape functions. The partition of unity imposed on mesh-free particle methods can be seen to be a generalization of shape functions for arbitrary integration point arrangements. From Li and Liu [14]...

...meshfree methods are the natural extension of finite element methods, they provide a perfect habitat for a more general and more appealing computational paradigm - the partition of unity.

The advantage of partition of unity methods is that any expression related to a field quantity can be imposed on the continuum. Where, for a bounded domain - in Euclidean space, a set of nonnegative compactly supported functions, sums to unity. Correspondingly, the value of some field function can be determined from its value at all other points via

\[ f(x) = \sum_{k=0,1,2,...n} a_k \phi(x - k) \]

The function \( f(x) \) can be related to any physical field expression; hydrodynamic, mechanical, electrical, chemical, magnetic etc. Such versatility is a key advantage of particle methods.

**Strategies for Programming on Multi-Core** By eliminating the need for ghost regions and high volume to surface ratio sub-domains, shared memory enables a near arbitrary selection of spatial divisions by which to distribute a numerical task. Correspondingly, it becomes convenient to repurpose a spatial division strategy already in use in one form or other in most particle based numerical codes. Spatial hashing techniques have been used widely to achieve algorithmically efficient interaction detection in particle codes (see for example DEM [15, 16]).

**SPATIAL HASHING IN PARTICLE METHODS**

The central idea behind spatial hashing is to overlay some regular spatial structure over the randomly positioned particles e.g. an array of equally sized cells. We can then perform spatial reasoning on the cells rather than on the particles themselves.
Spatial hashing assigns particles to cells or ‘bins’ based on a hash of particle coordinates. The numerical expense of such an algorithm is $O(N)$ where $N$ is particle number [15,16]. A variety of programmatic implementations of hash cell methods have been used (for example linked list). In this work a dictionary hash table is used where a generic list of particles is stored for each cell and indexed based on an integer unique key to that cell, i.e. $key = (k \times ny + j) \times nx + i$ where $nx$ and $ny$ are the total number of cells in the x and y dimensions and $i$, $j$ and $k$ are the integer cell coordinates in the x, y and z dimensions.

**Managing Thread Safety** The cells also provide a means for defining task packages to the cores. By assigning a number of cells to each core we assure “task orthogonality” in that each core is operating on different set of particles. Traditional software applications for shared memory parallel architectures have utilized locks to avoid thread contention. We note that a core may “read” the memory of particles belonging to surrounding cells but may not update them. We execute a single loop in which global memory for both previous and current field values ($V_{t_n}$ and $V_{t_{n+1}}$) is stored for each particle. Gradient terms can then be calculated as functions of values in previous memory, while updates are written to the current value memory, in the same loop. In parallel, minimizing the frequency of so called synchronization points has advantages for performance and we utilize this “rolling memory algorithm”. This allows the previous and updated terms to be maintained without needing to replace the former with the latter at the end of each step and thus, ensuring a single synchronization per step.

**Counter Intuitive Design** In a typical SPH simulator, two operations must be done on particles in each cell per step, separated by a synchronization, the first to determine the particle number density of each particle, and the second to perform the field variable updates. Interacting particles must be known for each of these two stages. Using a standard SPH formulation, the performance of two structure variations are compared in this case study: A. Interacting particles are determined in the first stage for all cells and stored in a global list for use in the second, and B. Interacting particles are determined as needed in each stage for each cell, i.e. twice per cell per time step. Recalculation of interacting particles means they need
only be kept in a local thread list that is overwritten with each newly dispatched cell. While differences in execution memory are to be expected of the two code versions, the differences in execution time are more surprising. For low core counts (< 10 cores), as would be expected, the single search variant (A) solves more quickly than the double (B) due to less computations. After this point, however, the double search (B) is shown to provide marked improvements in speed over (A) (up to 50%). This can be attributed to better cache blocking of the second approach and the significantly smaller amount of data experiencing latency when being loaded from RAM to cache. The fact that such performance gains only manifest when more than 10 cores are used, suggests that for less than 10 cores, RAM pipeline bandwidth is sufficient to handle a global interaction list.

![Fig 1. (a) memory vs. Number of Particles (b) Speed-up vs. Number of Cores](image)

The MIT SPH simulator (see [http://geonumerics.mit.edu](http://geonumerics.mit.edu)) has been validated for single and multi-phase flow. Figure 2 shows results for two phase Rayleigh-Taylor instability [17].
Fig 2. Rayleigh-Taylor instability example showing extremely complex fluid phase interface geometries; clear liquid is water, green liquid is oil. [9].

REFERENCES