Optimization of Parallel Particle-to-Grid Interpolation on Leading Multicore Platforms

Kamesh Madduri, *Member*, *IEEE*, Jimmy Su, Samuel Williams, Leonid Oliker, Stéphane Ethier, and Katherine Yelick, *Member*, *IEEE*

Abstract—We are now in the multicore revolution which is witnessing a rapid evolution of architectural designs due to power constraints and correspondingly limited microprocessor clock speeds. Understanding how to efficiently utilize these systems in the context of demanding numerical algorithms is an urgent challenge to meet the ever growing computational needs of high-end computing. In this work, we examine multicore parallel optimization of the particle-to-grid interpolation step in particle-mesh methods, an inherently complex optimization problem due to its low computation intensity, irregular data accesses, and potential fine-grained data hazards. Our evaluated kernels are derived from two important numerical computations: a biological simulation of the heart using the Immersed Boundary (IB) method, and a Gyrokinetic Particle-in-Cell (PIC)-based application for studying fusion plasma microturbulence. We develop several novel synchronization and grid decomposition schemes, as well as low-level optimization techniques to maximize performance on three modern multicore platforms: Intel's Xeon X5550 (Nehalem), AMD's Opteron 2356 (Barcelona), and Sun's UltraSparc T2+ (Niagara). Results show that our optimizations lead to significant performance improvements, achieving up to a $5.6\times$ speedup compared to the reference parallel implementation. Our work also provides valuable insight into the design of future autotuning frameworks for particle-to-grid interpolation on next-generation systems.

Index Terms—Particle mesh, particle-to-grid interpolation, multicore performance tuning, synchronization, atomic, lock free.

1 Introduction

Particle methods are used in a wide range of numerical simulations, including those from fusion physics, astrophysics, computational chemistry, and fluid mechanics. A key computational challenge in these simulations is the associated all-to-all particle interaction calculations. Luckily, in many of these domains, long range forces may be approximated, smoothed, or ignored, allowing scientists to replace the computationally expensive $O(N^2)$ methods with efficient $O(N)$ or $O(N \log N)$ approaches. Unfortunately, adoption of these approximations, while reducing the computational complexity of the problem, introduces potential data hazards in a parallel environment—thus, dramatically increasing the performance optimization challenges on memory-constrained multicore and manycore architectures.

In this work, we explore performance optimization of the $O(N)$ particle-mesh methods in the context of two diverse applications, a biological simulation of the heart using the Immersed Boundary (IB) method, and a Gyrokinetic Particle-in-Cell (PIC)-based method for studying plasma microturbulence. To facilitate our numerous optimization investigations, we create compact benchmarks for each application's most challenging phase: particle-to-grid interpolation. The particle-to-grid interpolation phase accounts for a substantial fraction—in some cases, up to 50 percent—of the overall running time in these two applications (see [1], [2], [3], [4] for a stepwise execution time breakdown of these applications at various scales of parallelism). Our experimental setup and the problem configurations studied are consistent with the overall distributed-memory parallelization scheme employed in these two applications, i.e., we have extracted the key single-node computational routine in these applications, and the speedup achieved with our new optimizations translates to a proportional overall application performance improvement [5].

Constrained by the desire for memory-efficient solutions and the need to avoid potential race conditions, we explore a number of partial replication and data-synchronization strategies in this work. While grid replication with ghost regions is a common optimization strategy to reduce communication volume in distributed memory implementations, our work focuses on efficient shared memory strategies. Further, we study fine-grained synchronization schemes that complement the replication strategies. To gauge the broad value of our techniques, we evaluate performance on three modern multicore systems: Intel's Xeon X5550 (Nehalem), AMD's Opteron 2356 (Barcelona), and Sun's UltraSparc T2+ (Niagara). Finally, to ensure the highest quality results, we employ a number of low-level optimizations designed to maximize performance on these Nonuniform Memory Access (NUMA) architectures.

- *K. Madduri is with The Pennsylvania State University, 343E IST Building, University Park, PA 16802. E-mail: madduri@cse.psu.edu.*
- *J. Su is with the University of California at Berkeley, 2390 Cresthaven Street, Milpitas, CA 95035. E-mail: jimmysu@eecs.berkeley.edu.*
- *S. Williams and L. Oliker are with Lawrence Berkeley National Laboratory. One Cyclotron Road, MS 50A-1148, Berkeley, CA 94720. E-mail: {SWWilliams, LOliker}@lbl.gov.*
- *S. Ethier is with Princeton Plasma Physics Laboratory, Princeton, NJ 08543. E-mail: ethier@pppl.gov.*
- *K. Yelick is with the University of California at Berkeley and Lawrence Berkeley National Laboratory, One Cyclotron Road, MS 50B-4230, Berkeley, CA 94720. E-mail: KAYelick@lbl.gov.*

Manuscript received 25 Oct. 2010; revised 12 Aug. 2011; accepted 30 Dec. 2011; published online 16 Jan. 2012.
Recommended for acceptance by A. Grama.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2010-10-0635.
Digital Object Identifier no. 10.1109/TPDS.2012.28.

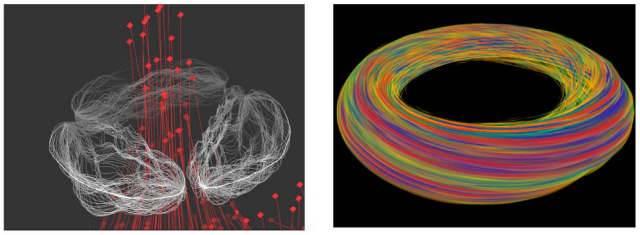
Our work makes the following new contributions:

- To the best of our knowledge, our work is the first to identify and generalize the particle-to-grid interpolation computation across the IB and gyrokinetic PIC methods, and present applicable novel multicore-tuned algorithms. Our proposed schemes are sufficiently generic to be applicable to a broad spectrum of particle-mesh-based applications.
- We explore the challenges of architecture-targeted optimization and memory-usage management in the context of complex data synchronization and irregular memory access patterns, and demonstrate that our methodology can achieve significant performance benefits on a variety of multicore platforms.
- We observe that the optimal implementation of the interpolation kernel depends both on application details and parallel platform characteristics. With a unique application-specific combination of optimizations, we achieve up to a $4\times$ and $5.6\times$ performance improvement for the heart spread force and PIC particle-to-grid phases, respectively.

2 Particle-Mesh Computations

Full long-range particle-particle force interaction simulations carry an unacceptably high computational complexity—$O(N^2)$ in the number of particles. Numerous techniques have therefore been developed to mitigate this force computation time. Particle-mesh computations are one such widely used class of methods which find applications in astrophysics [6], [7], plasma physics [8], computational chemistry, fluid mechanics, and biology. As the name suggests, the basic strategy adopted in particle-mesh methods is to utilize an auxiliary grid to approximate the density of particles as it varies in space and time. The key steps in a particle-mesh calculation include calculation of the charge density by assigning "charge" to the mesh points (referred to as the "particle-to-grid interpolation" step), solving the field potential equation (for instance, by solving Poisson's equation to calculate electrostatic or gravitational potential) on the mesh, calculation of force field based on the mesh potential, and then acceleration of particles based on the computed grid forces ("grid-to-particle interpolation" step). All the subroutines involving particles in a particle-mesh method have a computational complexity proportional to the total number of particles. This approach has the advantage of removing close encounter collision effects, thus allowing computation of the collective effects in a system with a smaller number of simulated particles.

Conceptually, the particle-to-grid interpolation step is analogous to generating a 3D histogram. However, unlike simple histograms in which items have integer coordinates and increments, particles located using floating point coordinates update a bounding box of grid points with a fraction of their charge (interpolation). Due to this indirection (a scatter add), randomly localized particles often suffer from poor cache reuse. The challenges of this deposition step become far more significant when one contemplates the Reduced Instruction Set Computing (RISC) nature of modern processors, as floating-point increment is typically not an atomic operation. This can prohibit or impede both intrathread and interthread parallelization of the particle array as multiple particles may attempt to update the same grid points simultaneously.



(a) (b)

Fig. 1. Advanced volume visualization of the Heart Code and GTC. (a) Heart tissue is modeled as elastic fiber (shown in gray) and exerts a force on the fluid it is immersed in (blood, depicted in red). (b) Electrostatic potential field created by the plasma particles in a GTC simulation.

We now present an overview of the two driver applications of our study, and compare their salient characteristics in the context of particle-to-grid interpolation. Sections 1 and 2 of the online supplementary document, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2012.28, present further details on the two applications.

2.1 Heart Code via Immersed Boundary Method

The immersed boundary method [9], [10] is a general technique for modeling elastic boundaries immersed within a viscous, incompressible fluid. This methodology has been applied to several biological and engineering systems, including the simulation of blood flow in the heart [10], sound waves in the cochlea [11], the swimming of eels, sperm and bacteria, and platelet aggregation during blood clotting [12]. These simulations have the potential to improve our basic understanding of biological systems and aid in the development of surgical treatments and prosthetic devices. While it is generally not categorized a particle-mesh method, we note that IB shares many similarities with that class of computations. In this paper, we study performance optimizations for a heart blood flow simulation code using the immersed boundary method. We will refer to this application in short as "Heart Code." Fig. 1a presents a graphic from our heart simulation code: the heart tissue is visualized as gray elastic fiber and exerts a force on the (red blood) fluid in which it is immersed. An important and expensive step in the Heart Code is the "spread force" calculation in which heart fiber forces are spread to the neighboring fluid cells. This is reminiscent of the particle-to-grid interpolation step in particle-mesh computations.

2.2 Plasma Simulation via Gyrokinetic PIC

Our second application driver for optimizing particle-to-grid interpolation is the Gyrokinetic Toroidal Code (GTC) [13], [14]. GTC was developed to study the global influence of microturbulence on particle and energy confinement, critical elements in understanding practical fusion power plant designs. It is a 3D, fully self-consistent PIC code which solves the kinetic equation in a toroidal geometry. The charge update scheme in GTC differs from traditional PIC codes. In the classic PIC method, a particle is followed directly and the charge is distributed to its nearest neighboring grid points. However, in the gyrokinetic PIC approach used in GTC, the fast circular motion of the

TABLE 1
Overview of Evaluated Multicore SMPs

Core Architecture	AMD Barcelona	Intel Nehalem	Sun Niagara2
Type	superscalar out-of-order	superscalar out-of-order	multithreaded dual-issue
Clock (GHz)	2.30	2.66	1.16
DP GFlop/s	9.2	10.7	1.16
L1 Data Cache	64 KB	32 KB	8 KB
private L2 cache	512 KB	256 KB	—
System	**Opteron 2356 (Barcelona)**	**Xeon X5550 (Gainestown)**	**UltraSparc T5140 (Niagara)**
sockets	2	2	2
cores (threads) per socket	4 (4)	4 (8)	8 (64)
Memory Parallelism	HW prefetch	HW prefetch	Multithreading
Shared last-level caches	2×2 MB†	2×8 MB†	2×4 MB†
DRAM Capacity	16 GB	12 GB	32 GB
DRAM Pin Bandwidth (GB/s)	21.33	51.2	42.66(read) 21.33(write)
DP GFlop/s	73.6	85.3	18.7

†*One per socket.*

TABLE 2
Summary of Heart Code and GTC Problem Instances

Heart Spread Force	Small (S)	Medium (M)	Large(L)
Grid Dimensions	256^3	256^3	256^3
Heart Radius	25	50	100
total grid points	16,777,216	16,777,216	16,777,216
grid points touched	31,500	125,700	502,500
particles per (cubical) point	*— total number of particles —*		
0.19	3.2M	3.2M	3.2M
0.95	16M	16M	16M
4.77	80M	80M	80M

GTC charge deposition	Small (S)	Medium (M)	Large(L)
mzeta	1(+ghost)	1(+ghost)	1(+ghost)
mpsi	192	384	768
mthetamax	1408	2816	5632
mgrid	151,161	602,695	2,406,883
total grid points	302,322	1,205,390	4,813,766
total grid points touched	302,322	1,205,390	4,813,766
micell (#particles/mgrid)	*— total number of particles —*		
2	0.3M	1.2M	4.8M
10	1.5M	6.0M	24.0M
50	7.6M	30.0M	120.0M

charged particle around the magnetic field lines is averaged out and replaced by a charged ring. To study low frequency instabilities that affect energy transport, GTC uses four points on the charged ring [15] to describe the nonlocal influence of the particle orbit. In this way, the full influence of the fast, circular trajectory is preserved without having to resolve it. However, this scheme inhibits straightforward shared-memory parallelism even further, since the updated positions need to be computed for each particle and at each time step. Furthermore, multiple particles may concurrently attempt to deposit their charge onto the same grid point. Fig. 1b shows the 3D visualization of electrostatic potential in global, self-consistent GTC simulation of plasma microturbulence in a magnetic fusion device.

3 Experimental Setup

We explore optimizations for the two applications across a variety of leading multicore platforms, summarized in Table 1. To mitigate the impact of long latency snoopy coherency protocols, we limit ourselves to the dual-socket Symmetric Multiprocessors (SMPs) of the AMD Barcelona, Intel Nehalem, and Sun Niagara2 systems. In the near future, as the number of cores per socket continues to scale, we expect on-chip coherency to remain manageable and bounded. However, as the number of sockets scale, the snoopy protocol can become a performance impediment. Thus, we expect dual-socket multicore SMPs to be reasonable proxies for future multicore systems. Additional details of the evaluated platforms are presented in Section 3 of the online supplementary document.

3.1 Overview of Problem Instances

For our examined applications, we extract the key performance bottlenecks—particle-to-grid interpolation—and encapsulate them into standalone benchmarks. A summary of important problem instances is presented in Table 2. Additional details of performance expectations and setup are given in Section 4 of the online supplementary document.

3.1.1 Heart Code Kernel

The Heart Code application is a shared memory threaded code based on an earlier distributed-memory version [3]. To create a standalone Heart Code benchmark, we extracted its most challenging kernel, the spread force calculation, which is essentially a particle-to-grid interpolation.

The three input parameters in the Heart Code, used to describe the test simulation are the dimensions of the cubical grid (i.e., resolution in X, Y, and Z), the radius of the heart, and the number of heart fibers (particles). These are summarized in Table 2. We fix the dimensions of the fluid grid to be 256^3, vary the average density of particles (number of particles/cubical grid points) from 0.19 to 4.77, and the radius of the heart from 25 to 100; the latter is simply labeled as *small, medium, and large*. Particles (heart fibers) are randomly distributed on the surface of the heart sphere such that each octant has the same number of particles, and the density on the surface of the sphere is sufficiently high for the underlying physical simulation. These parameters represent perturbations to commonly used configurations.

3.1.2 GTC Kernel

As with the Heart Code, GTC's parallelization of the charge deposition (particle-to-grid interpolation) is the most challenging step. To that end, we extract the GTC charge deposition kernel. For the threaded implementation, all MPI (reference implementation) calls are removed in favor of shared memory accesses. Similar to the Heart Code, we evaluate the particle-to-grid interpolation phase, and as such, the benchmark only loops over this one kernel.

There are several input parameters in GTC to describe the test simulation, summarized in Table 2. The ones most relevant to the charge deposition kernel are the size of the discretized toroidal grid, the total number of particles, and the Larmor radius distribution of the particles for the four-point gyrokinetic averaging. Often one replaces the number

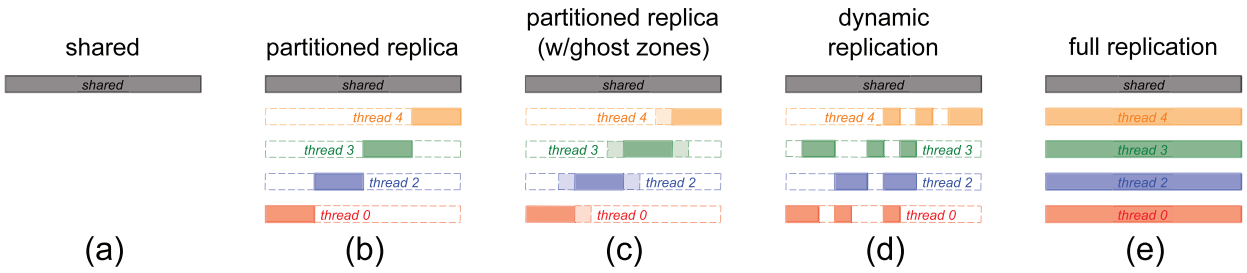


Fig. 2. The five grid decomposition/replication strategies we utilized in the Pthreads implementations. Note, color denotes thread-private data. As the replication in (d) is particle-distribution dependent, we only show an arbitrary replication pattern.

of particles with the average particle density as measured in the ratio of particles to grid points (labeled as *micell*). Three coordinates describe the position of a particle within the discretized torus: ζ (*zeta*, the position in the toroidal direction), ψ (*psi*, the radial position within a poloidal plane), and θ (*theta*, the position in the poloidal direction within a toroidal slice). The corresponding dimensions of a toroidal grid segment are: *mzeta*, *mpsi*, and *mthetamax*. In this paper, we explore three different grid problem sizes, labeled *small, medium, and large*, and experiment with average particle densities of 2, 10, and 50 particles per grid point. Particles are uniformly randomly distributed throughput the grid (unlike the Heart Code where particles are constrained to a spherical subset of the grid). Note that, these settings are similar to the ones used in prior experimental studies and GTC production runs [1], [2].

4 Particle-to-Grid Optimization

In this section, we outline the key synchronization and replication techniques. Application-specific details and further low-level tuning details are discussed in Sections 5 and 6 of the online supplement.

4.1 Synchronization Optimizations

In traditional MPI implementations of PIC methods, such as the original GTC version, the density grid is fully replicated among processes, allowing autonomous updates of local copies without potential conflicts. MPI communication then proceeds to reduce all these grid copies into one, while redistributing the data for the solve phase. Unfortunately, such approaches are wasteful with respect to increasingly power-hungry DRAM. Luckily, modern multicore SMPs allow us to trade multiple processes per SMP for multiple threads per process; thus eliminating redundant grid copies for one copy in shared memory. However, the particle-to-grid interpolation includes a scatter-increment operation, creating potential data hazards when multiple threads may attempt to simultaneously increment the same memory location.

One obvious solution to this problem is to implement a lock for each element requiring synchronized access, referred to here as fine-grained locks. This approach requires one lock for each grid point (a 64-bit double-precision number). Clearly, fine-grained locks involve both a substantial computational and memory overhead, as a lock must be acquired and released for every grid update. However, the cost of these updates can be amortized by assigning one lock to multiple grid elements. As there is a substantial 3D spatial correlation between grid updates in both Heart Code and GTC, we define two additional locking strategies for each that vary the number of grid points locked at a time. These are referred to as medium and coarse-grained locks. The details of how neighboring grid cells are grouped for a given lock are application-dependent, and are discussed in the online supplement.

Our final solution for the synchronization problem is to implement atomic floating-point increment via the atomic Compare and Swap (CAS) instruction. On the x86 machines, we wrote hand-crafted assembly routine to accomplish this, while on the SPARC architecture, we utilize the CAS intrinsic.

4.2 Partial Replication Optimizations

Although exploration of various synchronization strategies allows some freedom to amortize the overhead associated with updates to the shared grid, we explore additional amortization via grid replication. This approach makes subsets of a grid replica thread private, where a given thread may update certain (privately owned) grid points in the replica without synchronization. Doing so allows the reducing synchronization, in exchange for global reduction overhead at the conclusion of the updates. Additionally, there is potential for locality improvement, as private data reduces the volume of thrashing between the cores within a socket, or between the sockets of an SMP.

We define five, geometrically based, grid replication strategies. Their stylized visualizations are shown in Fig. 2, where a high-dimensional grid has been reduced to a 1D box. The solid portions of the box represent the subsets of the grid "owned" by that thread. Reading the figure from left to right shows a reduction in locking overhead, but an increase in memory usage combined with more expensive reductions. In the rest of this section, we discuss the five approaches and detail how we map these generic strategies to the requirements of the Heart Code and GTC.

Shared grid. In the baseline shared grid approach, there is one copy of the grid shared by all threads. Selecting the appropriate synchronization approach must therefore be handled carefully. Note that, if each thread's particles are spatially binned, then each thread's cache will be mapped to portion of this array. However, there can be thrashing at the boundaries, as multiple threads may contend for the data. Since there is just one copy of the grid, there is no need for a reduction.

Partitioned replica. When particles are spatially binned, we can create one replica of the grid that is statically and geometrically partitioned among threads (based on the distribution of particles among threads). Doing so allows

the elimination of virtually all locks in exchange for updates to the replica (based on a simple inspection of a particle's coordinate). There are a few cases at the boundaries of these partitions where a thread will need to update a grid location outside its partition. As it may not access another thread's replica (due to the lack of reciprocal synchronization), the thread must instead update the shared copy via an appropriate synchronization mechanism. Once all P threads have completed their updates and pass through a barrier, all the data in the P partial replicas is added to the shared grid. Although, this method doubles the grid memory capacity requirements and uses a simple reduction, it eliminates a substantial fraction of the synchronization operations for moderate thread parallelism (the savings are application specific).

Partitioned replica (with ghost zones). To further eliminate synchronizations at the boundaries, a ghost region can be added to each thread's portion of the replica. The ghost region consists of the expected spatial overlap resulting from the grid points updated by a particle. This virtually eliminates all synchronization traffic, but requires additional space.

Dynamic replication. When particle distributions are extremely nonuniform, the inflexibility of the static partition grid methods results in large reductions of zeros; such an approach is inefficient and should be avoided. To that end, we also propose a dynamic replication strategy in which on the first step, threads determine the points they will update and allocate 3D blocks from a pool of memory as replicas. Threads may subsequently update their private replica, and once done, must reduce these sparsely populated replicas with the shared grid. Such an approach completely eliminates the need for synchronized accesses to the shared grid, but requires complex indirection and a reduction at phase completion.

Full replication. As a final comparison point, we include a full replication strategy. Here, every thread replicates the entire grid and thus mimics the MPI algorithm, exchanging message passing for loads and stores to shared memory. This approach eliminates all synchronization (thus our fifth synchronization strategy: "none"), but requires copious DRAM memory and large reductions. Such strategies are acceptable at extremely high particle densities (when the particles per grid point is greater than the number of threads) or when the overhead of any synchronization is prohibitively expensive.

Table 3 presents the replication, synchronization, and low-level tuning optimizations employed by each kernel. Of the 25 possible choices, only the relevant combinations of synchronization and data replication are chosen.

TABLE 3
Optimizations Implemented by Kernel

		Heart Code					GTC				
		synchronization					synchronization				
		coarse	medium	fine	atomics	none	coarse	medium	fine	atomics	none
Replication	None (Shared)	✓	✓	✓	✓		✓	✓	✓	✓	
	Partitioned	✓	✓	✓	✓		✓	✓	✓	✓	
	Partitioned+Ghost	–	–	–	–		✓	✓	✓	✓	
	Dynamic	✓	✓	✓	✓		–	–	–	–	
	Full					✓					✓

		Heart Code	GTC
Miscellaneous	Particle Sorting	✓	✓
	Thread Affinity	✓	✓
	NUMA Allocation	✓	✓
	Data Layout	✓	✓
	Loop Fusion	–	✓
	SIMD	–	x86-only

5 Performance Results and Analysis

To provide analysis via extraction of salient performance bottlenecks, we vary problem size, particle density, approaches to data synchronization, and data replication, as well as thread-level parallelism.

5.1 Synchronization and Replication Analysis

We begin by presenting the tradeoffs between synchronization (locks and atomics) and data replication in the context of GTC. Fig. 4 shows performance as a function of synchronization strategy—**(C)**oarse, **(M)**edium, **(F)**ine, **(A)**tomics—for each replication strategy on Nehalem; recall that full replication does not require synchronization (Table 3). Note that, experiments conducted to evaluate the tradeoffs between locks and atomics on the Heart Code (not shown) determined that atomics always delivered significantly better performance.

Across all grid replication strategies, performance improves as the locking overhead is amortized (Fine to Coarse) with an additional benefit (Fine versus Atomics) seen when amortizing Pthread's fine-grained locking overhead by direct use of atomics. Observe that at full parallelism on Nehalem (16-way threading), contention for locks does not impede performance. Additionally, our partitioned replica with ghost zones replication in conjunction with atomic updates, achieves performance on par with the naïve full replication strategy. Although the code is more complex than the simpler full replication, our approach not only eliminates a global reduction, but dramatically reduces the grid memory requirements from $O(P \cdot mgrid)$ to $O(mgrid)$ (where P is the number of threads of execution). In GTC, a vast amount of computation is required to determine grid array indices and interpolate charge; thus the overhead of atomic updates is effectively amortized. As this ratio is unique to GTC, it is appropriate for all codes to explore the best synchronization and replication strategy.

We observe that the benefits of affinity and SIMDization are relatively low on the GTC kernel-architecture combination. Given that the baseline approach includes NUMA-aware allocation optimization, this ensures that grid and particle arrays are evenly distributed across memory banks. The small benefit with affinity pinning seems to indicate that the OS default thread pinning for this case may be similar to our thread pinning strategy. SIMD is of little benefit, as the code requires the streaming through a few million particles and is memory-bound for reasonable cache locality of grid points.

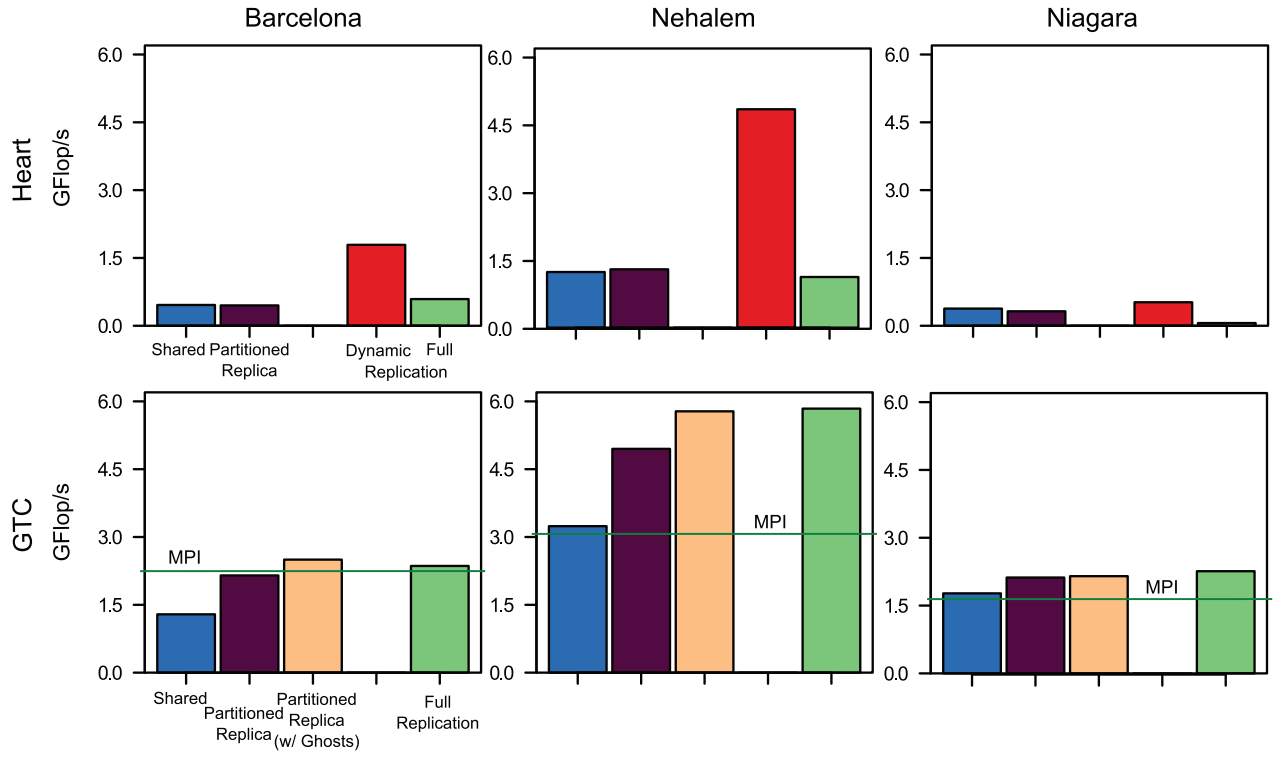


Fig. 3. Performance relative to various grid replication strategies for the medium grid size and medium particle density problem.

5.2 Replication Parallel Performance Analysis

For each kernel, we specialize the five general grid replication strategies and apply them to our two codes. The Heart Code does not include the *replica with ghost* approach as particles are spatially confined, while GTC does not include *dynamic replication* as the particle distribution is uniform. Fig. 3 presents the results on the three evaluated architectures for the medium sized grid with (on average) moderate particle density (approximately one per grid point for Heart, and 10 per grid point for GTC). For each strategy, we present parallel performance using the best synchronization method as well as all relevant optimizations.

On the Heart Code, we observe that the shared and partitioned replica approaches deliver comparable performance. This may seem surprising, as the partitioned replica should eliminate most atomic accesses. However, given the bulk synchronous nature of the replica algorithm, performance is limited to the slowest thread (as all threads have equal numbers of particles, the one with the most updates of the shared grid). For any moderate degree of parallelism, at least one thread's updates are entirely to the shared grid. Thus, it is no faster than the naïve shared implementation. The full replication approach wastes (linearly with threads) flops, bandwidth, and memory capacity in order to avoid all synchronization. This results in comparable performance on low concurrency x86 processors, but very low performance on highly multithreaded processors such as Niagara where the wasted flops and bandwidth from the reduction dominate. Clearly, the efficient dynamic replication provides the best of both worlds by eliminating most synchronization without wasting flops, bandwidth or memory capacity (for nonuniform distributions).

On GTC, observe that strategies which provide some replication attain nearly twice the performance of the reference shared grid approach (on all three platforms). This demonstrates that replacing synchronized updates (locks or atomics) with regular loads and stores has a substantial benefit for the typical case. Additionally, we observe that there is no substantive performance benefit of using the memory hungry full replication strategy. The green line represents the performance of the MPI implementation (algorithmically closest to the full replication strategy) of the same problem configuration. On Barcelona,

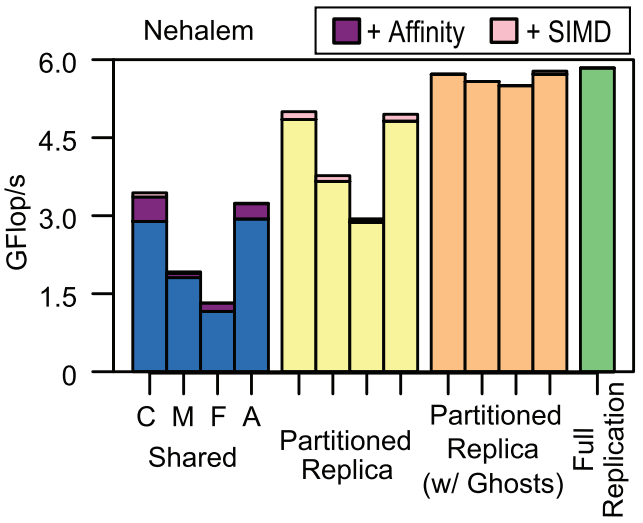


Fig. 4. GTC parallel performance on Nehalem relative to varying replication and synchronization strategies using **(C)**oarse, **(M)**edium, **(F)**ine, and **(A)**tomics, for the M10 problem size. Results also show the impact of affinity and SIMD optimizations.

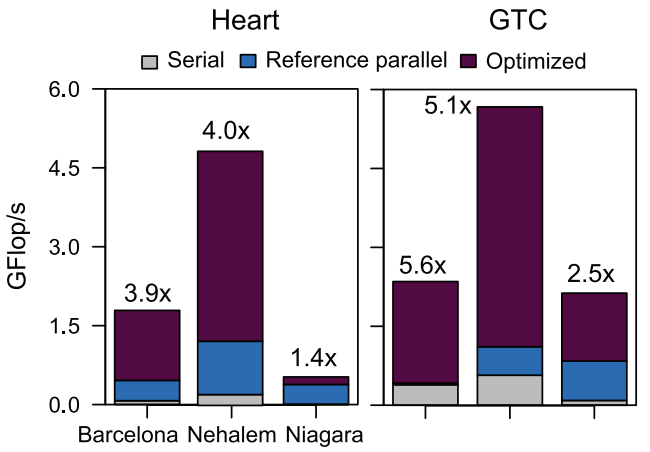


Fig. 5. Summary performance impact of the various tuning strategies. Speedup numbers indicate the performance ratio of optimized implementation to the reference parallel code.

observe that the performance of both the Pthreads and MPI schemes are quite similar. On Nehalem, no improvement was achieved when spawning 16 processes, indicating that the MPI implementation was unable to utilize the two-way SMT support. Similarly, we were unable to spawn more than 64 MPI processes on the Niagara system, and this may explain the performance gap between the shared memory full replication version and the MPI implementation. Further, our memory efficient version outperforms the MPI code, while only requiring a tiny fraction of the memory capacity. This difference is most profound on Niagara, where the memory for shared grids is roughly only 1 percent compared with MPI's distributed and replicated grids.

Overall results show that atomics are slow but useful in avoiding superfluous replication. Thus, the grid replication strategy must be judiciously chosen based on particle distribution and platform. Note also that machine performance is largely correlated with its bandwidth. Additional experimental details highlighting scalability analysis and problem configuration perturbations are presented in Section 7 of the online supplementary document.

5.3 Cross-Architecture Comparison

Fig. 5 presents summary performance after parallelization and optimization. The reference Heart Code computation corresponds to the shared grid version with particle sorting and atomic updates, while GTC's reference parallel version uses a shared grid with fine-grained locking and NUMA-aware memory allocation.

Observe that on most machines, parallelism significantly improves performance. Additionally, optimization (via synchronization and replication) delivers a further boost to parallel performance. Overall our optimized implementation outperformed the reference serial case by up to $87.5\times$ and $23.6\times$, and the reference parallel case by up to $4\times$ and $5.6\times$, for the Heart Code and GTC, respectively. Additionally, Nehalem achieves the best performance as it has relatively superior flops capability, bandwidth rate, and cache size. Barcelona delivers better performance than Niagara on the compute-intensive Heart Code while achieving comparable runtimes on the more bandwidth-intensive GTC.

6 Conclusions

In this work, we explore a number of techniques designed to improve particle-to-grid interpolation—the most challenging phase of particle-in-cell computations—on modern multicore systems. Rather than the accepted flat replication strategy found in MPI implementations, we exploit the trend toward large manycore processors by selecting and optimizing an implementation in which the target grid is shared and with minimized replication.

Results show that the optimal implementation and parameterization was unique to a particular application, target architecture, and particle distribution. For example, we found that in the Heart Code, the nonuniform distribution of heart fibers within the fluid grid demanded dynamic replication to minimize accesses to the shared grid and the complexity of the final reduction. Conversely, the spatially uniform distribution of particles coupled with nonlocalized grid updates arising from the Larmor radius pushed GTC toward the partitioned grid with ghost zones approach. Overall, we were able to reduce memory usage substantially while simultaneously significantly improving performance via our novel optimization schemes.

In future, we will continue to generalize our approach and encapsulate it behind a single interface that hides the complexity of performance optimization and tuning from the application scientist. Additionally, we will extend this concept to the distributed memory PGAS environment. Finally, the optimizations and analyses presented here are a critical step toward designing autotuning frameworks, which will be extended to numerous classes of particle methods and architectural platforms.

Acknowledgments

All authors from Lawrence Berkeley National Laboratory were supported by the DOE Office of Advanced Scientific Computing Research under contract number DE-AC02-05CH11231. Dr. Ethier was supported by the DOE Office of Fusion Energy Sciences under contract number DE-AC02-09CH11466. Additional support comes from Microsoft (Award #024263), Intel (Award #024894), U.C. Discovery (Award #DIG07-10227), as well as Par Lab affiliates, including National Instruments, NEC, Nokia, NVIDIA, Samsung, and Sun Microsystems.

References

[1] M. Adams, S. Ethier, and N. Wichmann, "Performance of Particle in Cell Methods on Highly Concurrent Computational Architectures," *J. Physics: Conf. Series*, vol. 78, p. 012001, 2007.

[2] S. Ethier, W. Tang, R. Walkup, and L. Oliker, "Large-Scale Gyrokinetic Particle Simulation of Microturbulence in Magnetically Confined Fusion Plasmas," *IBM J. Research and Development*, vol. 52, nos. 1/2, pp. 105-115, 2008.

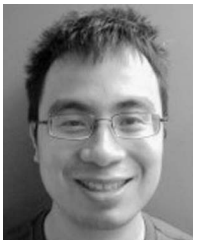
[3] E. Givelberg and K. Yelick, "Distributed Immersed Boundary Simulation in Titanium," *SIAM J. Scientific Computing*, vol. 28, no. 4, pp. 1361-1378, 2007.

[4] J. Su, "Optimizing Irregular Data Accesses for Cluster and Multicore Architectures," PhD dissertation, Univ. of California, Berkeley, http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-170.html, Dec. 2010.

[5] K. Madduri, K. Ibrahim, S. Williams, E.-J. Im, S. Ethier, J. Shalf, and L. Oliker, "Gyrokinetic Toroidal Simulations on Leading Multi- and Manycore Hpc Systems," *Proc. ACM/IEEE Conf. Supercomputing (SC '11)*, pp. 1-11, Nov. 2011.

[6] R. Hockney and J. Eastwood, *Computer Simulation Using Particles.* Taylor & Francis, Inc., 1988.
[7] E. Bertschinger and J. Gelb, "Cosmological N-Body Simulations," *Computers in Physics,* vol. 5, pp. 164-175, 1991.
[8] C. Birdsall and A. Langdon, *Plasma Physics via Computer Simulation.* McGraw Hill Higher Education, 1984.
[9] R. Mittal and G. Iaccarino, "Immersed Boundary Methods," *Ann. Rev. Fluid Mechanics,* vol. 37, pp. 239-261, 2005.
[10] C. Peskin and D. McQueen, "A Three-Dimensional Computational Method for Blood Flow in the Heart: (I) Immersed Elastic Fibers in a Viscous Incompressible Fluid," *J. Computational Physics,* vol. 81, pp. 372-405, 1989.
[11] R. Beyer, "A Computational Model of the Cochlea Using the Immersed Boundary Method," *J. Computational Physics,* vol. 98, pp. 145-162, 1992.
[12] D. McQueen and C. Peskin, "A Three-Dimensional Computer Model of the Human Heart for Studying Cardiac Fluid Dynamics," *Computer Graphics,* vol. 34, pp. 56-60, 2000.
[13] Z. Lin, T. Hahm, W. Lee, W. Tang, and R. White, "Turbulent Transport Reduction by Zonal Flows: Massively Parallel Simulations," *Science,* vol. 281, no. 5384, pp. 1835-1837, 1998.
[14] S. Ethier, W. Tang, and Z. Lin, "Gyrokinetic Particle-in-Cell Simulations of Plasma Microturbulence on Advanced Computing Platforms," *J. Physics: Conf. Series,* vol. 16, pp. 1-15, 2005.
[15] W. Lee, "Gyrokinetic Particle Simulation Model," *J. Computational Physics,* vol. 72, no. 1, pp. 243-269, 1987.

Kamesh Madduri received the bachelor degree from the Indian Institute of Technology, Madras, and the PhD degree from the College of Computing at Georgia Institute of Technology. He is an assistant professor in the Computer Science and Engineering Department at The Pennsylvania State University. His research interests include combinatorial scientific computing, multicore performance tuning of scientific applications, and scientific data analytics. He is a member of the IEEE.



Jimmy Su received the PhD degree from Berkeley in 2010. He was a graduate student researcher in the Computer Science Department at the University of California at Berkeley. His research interests include interaction between parallel computing, compilers, and the titanium data parallel language.

Samuel Williams received the bachelor degree in electrical engineering, mathematics, and physics from Southern Methodist University, and the master and PhD degrees in computer science from the University of California at Berkeley. He is a computer scientist in the Future Technologies Group at the Lawrence Berkeley National Laboratory. His current research interests include performance and energy optimization on multicore architectures.

Leonid Oliker received the bachelor degree in computer engineering and finance from the University of Pennsylvania, and performed both the doctoral and postdoctoral work at NASA Ames research center. He is a senior computer scientist in the Future Technologies Group at Lawrence Berkeley National Laboratory. He has coauthored more than 90 technical articles, five of which received best paper awards. His research interests include HPC evaluation, multicore autotuning, and power-efficient computing.

Stéphane Ethier received the PhD degree from the Department of Energy and Materials of the Institut National de la Recherche Scientique in Montreal, Canada. He is a computational physicist in the Computational Plasma Physics Group at Princeton Plasma Physics Laboratory (PPPL). His current research interests include high-performance computing and large-scale gyrokinetic particle-in-cell simulations of microturbulence in magnetic connement fusion devices.

Katherine Yelick received the bachelor's, master's, and PhD degrees from the Massachusetts Institute of Technology. She is a professor of electrical engineering and computer sciences at the University of California at Berkeley and associate laboratory director for computing sciences at Lawrence Berkeley National Laboratory. She has led or coled the Berkeley UPC, Titanium, and Bebop projects. Her current research interests include parallel computing, memory hierarchy optimizations, programming languages, and compilers. She is a member of the IEEE.