Extending Stability Beyond CPU Millennium

A Micron-Scale Atomistic Simulation of Kelvin-Helmholtz Instability

J. N. Glosli
Lawrence Livermore National Laboratory Livermore, CA

K. J. Caspersen
Lawrence Livermore National Laboratory Livermore, CA

J. A. Gunnels
IBM Corporation Yorktown Heights, New York USA

D. F. Richards
Lawrence Livermore National Laboratory Livermore, CA

R. E. Rudd
Lawrence Livermore National Laboratory Livermore, CA

F. H. Streitz
Lawrence Livermore National Laboratory Livermore, CA

ABSTRACT

We report the computational advances that have enabled the first micron-scale simulation of a Kelvin-Helmholtz (KH) instability using molecular dynamics (MD). The advances are in three key areas for massively parallel computation such as on BlueGene/L (BG/L): fault tolerance, application kernel optimization, and distribution across the parallel architecture. In particular, we have developed novel capabilities for handling hardware parity errors and improved particle-based domain decomposition algorithms to minimize communication overhead, achieve excellent scalability, and improve overall application performance. We have also extended the ddcMD code to handle commonly used Embedded Atom Method (EAM) interatomic potentials efficiently. As a result we have conducted a 2 billion atom KH simulation amounting to 2.8 CPU-millennia of run time, including a single, continuous simulation run in excess of 1.5 CPU-millennia. The current optimized ddcMD code is benchmarked at 54.3 TFlop/s with the EAM potential, with additional improvements ongoing. These improvements enabled us to run the first MD simulation of a micron-scale system developing a KH instability.

1. INTRODUCTION

With 131,072 CPUs, BlueGene/L (BG/L) at Lawrence Livermore National Laboratory is the first and so far the only supercomputer in the world to employ over 100,000 processors, holding first place on the Top-500 list [38]. Achieving the highest levels of performance using this large number of processors requires not only a well optimized application kernel, but also truly scalable solutions to issues such as communications overhead, load imbalance, I/O, and redundant computation. Techniques that are perfectly adequate for 1,000 or even 10,000 CPUs don't always perform as expected on 100,000 CPUs. However, scalability is only part of the challenge of working on BG/L. With such a large number of processors—at least 10 times more than almost every other current supercomputer—hardware failures are a virtual certainty during any substantial job. Without a well designed recovery strategy hardware failures can substantially impact performance. This situation is currently unique to BG/L, but it is sure to be encountered with increasing regularity as chips with 10's or even 100's of cores are used to build future supercomputers with millions of CPUs.

Traditionally, hardware errors have been handled either by the hardware itself or the by operating system. However, a greater degree of robustness and flexibility can be attained by allowing the application to participate in the error correcting (handling) process. The application, with its understanding of the details of the calculation, can evaluate the impact of the error and decide the most efficient strategy for recovery. This software capability opens the potential for a new paradigm in supercomputer design. Relaxed hardware reliability constraints made possible by application-assisted error recovery open the door to designs using less intrinsically stable but higher performing or perhaps less expensive components thus improving the price/performance ratio. To compare the effectiveness of these error recovery techniques we discuss long-running large-length-scale MD simulations [1] of hydrodynamic phenomena (in particular the Kelvin-Helmholtz instability) with atomistic resolution.

The Kelvin-Helmholtz (KH) instability [6, 2] arises at the interface of fluids in shear flow, and results in the formation of waves and vortices. Waves formed by KH instabilities are ubiquitous in nature. Examples include waves on a windblown ocean or sand dune, swirling cloud billows (see Fig. 1), and the vortices of the Great Red Spot and other storms in Jupiter's atmosphere. The KH instability has been studied previously by a variety of means, but never before using molecular dynamics with realistic atoms. The growth of the instability in the linear regime has been studied analytically based on the Navier-Stokes equation [22]. Beyond linear

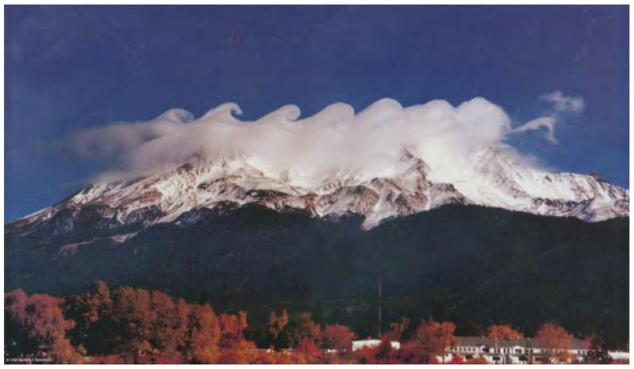

Figure 1: Clouds over Mount Shasta exhibiting a Kelvin-Helmholtz instability [34].

Supercomputing '07 Nov. 2007, Reno, NV UCRL-CONF-230679
Copyright 2007 ACM 978-1-59593-764-3/07/0011 ...$5.00.

analysis, the phenomenon has been studied numerically using techniques including lattice Boltzmann [41], Smooth Particle Hydrodynamics [19, 24], and Direct Simulation Monte Carlo [39] and other hard-sphere particle techniques, as well as Navier-Stokes [37, 11, 28, 10, 9]. Hydrodynamic instabilities related to the KH instability have been studied with MD such as the shedding of vortices from cylinders in a flowing fluid [29], interface roughening in sandpiles [3], and the Rayleigh-Taylor instability in which the mushrooming of the plumes is related to the KH instability [20].

We simulated KH initiation and development at the interface between two molten metals in a sample of over 2 billion atoms via molecular dynamics. The use of MD in our work has enabled simulation of fluids where all phenomena of interest—vortex development, species interdiffusion, interface tension, and so on—are fully consistent, arising from a single interatomic force law.

In order to achieve extended high performance on massively parallel computers one needs: superior processor utilization, efficient distribution of tasks, and long-term stability without performance cost. In the rest of the paper we address these needs with: an efficient implementation of a standard interatomic potential, a robust particle-based domain decomposition strategy, and an implementation of application error management.

The paper is organized as follows: In section 2 we discuss the creation of hardware-fault tolerant molecular dynamics; In section 3 we describe kernel optimization strategies; In section 4 we present performance results via scaling, benchmarks, and the early stages of an ongoing simulation; In section 5 we discuss science results obtained from a completed exploratory simulation[1]; Finally, in section 6 we present our conclusion.

2. HARDWARE-FAULT TOLERANT MD

Micron-scale MD simulation requires the massively-parallel architecture of machines such as BG/L, as well as adaptable software such as ddcMD that can exploit the novel architecture. BlueGene/L is a massively-parallel scientific computing system developed by IBM in partnership with the Advanced Simulation and Computing program (ASC) of the US Department of Energy's National Nuclear Security Agency (NNSA). BG/L's high-density cellular design gives very high performance with low cost, power, and cooling requirements. The 65,536-node (131,072-CPU) system at LLNL is at this writing the fastest supercomputer in the world, having achieved a performance of 280.6 TFlop/s on the Linpack benchmark in November, 2005.

Although system designers have spent considerable effort to maximize mean time between failures (MTBF), the large number processors on BG/L greatly increases the likelihood of unrecoverable hardware errors. Error rates that would be unnoticeable in serial computing can become crippling problems. A component that produces errors at the rate of once every 2 or 3 years on a typical desktop machine will cause a failure on average every 10 minutes on BG/L.

Many techniques to improve or address hardware fault tolerance have been described and/or implemented. For example the well known SETI@HOME project [33] uses a job farming/redundant calculation model to detect and correct faults. Other redundant calculation schemes employ multiple threads on different cores [13], compiler inserted redundant instructions [30, 31], and even systems with multiple redundant processors and data buses [40] just to list a few. Checkpoint or log based rollback schemes [7] offer an alternative to redundancy. Although these techniques all attempt to mitigate errors at the hardware or system software level, some applications can also be made fault tolerant through the selection of an appropriate underlying algorithm [8].

The primary hardware failure mode on BG/L has been transient parity errors on the L1 cache line. Although most components of the BG/L memory subsystem can correct single bit parity errors and detect double bit errors, the L1 cache can only detect single bit errors and is unable to correct them. When running on full BG/L, L1 parity errors occur on average every 8 hours, or approximately once every CPU century. Without assistance from the application, the Compute Node Kernel (CNK) can only terminate the job and force a reboot of the machine. At the system level where one corrupted bit is equally important as any other, there is no other viable strategy that can guarantee the fidelity of a calculation.

Recovery from a termination is expensive: 1/4 hour to reboot the machine, 1/4 hour to restart the application from the last checkpoint, and on average perhaps 1 hour (half the checkpoint interval) to redo calculations since the last checkpoint. Hence 1.5 out of every 8 hours, or nearly 20% of (wall-clock) time is spent in error recovery. Applications with longer checkpoint intervals will suffer even greater losses. More frequent checkpoints could reduce the recovery time, but the time spent writing could easily offset the advantage.

Since supercomputing applications often require days of run time the error recovery time represents a significant reduction in the overall computational efficiency. In evaluating the overall performance of an application the time spent recovering from errors must be factored in. Computer time is not budgeted in Flop/s but in wall clock or CPU hours—time spent either down or repeating calculations lost to a crash increases the project budget.

2.1 Parity Error Recovery Methods

BG/L provides two methods to mitigate the impact of L1 parity errors. The first option is to write through the L1 data cache directly to lower levels of the memory subsystem. Using write-through mode eliminates the possibility for unrecoverable parity errors (data can always be reloaded from main memory in the case of an error) but at the cost of degraded performance. The second option transfers control to an application-supplied interrupt handler whenever the CNK detects an unrecoverable parity error, thus allowing the application to assist in the error recovery process. By exploiting detailed knowledge of the application state and/or memory usage, the application can use this handler to implement highly efficient recovery strategies that would otherwise be impossible.

Activating write-through mode is very effective at eliminating parity errors. One of the jobs that generated the data reported in Section 5 ran without interruption for more than 4 days and logged 12 recovered parity errors. This run represents over 1.5 CPU millennia without an unrecoverable error. Unfortunately, the increase in stability comes with a performance cost. The cost of write-through mode is application dependent with performance decreases typically in the range of 20–50%. For ddcMD the performance degradation caused by activating writethrough mode is 20%—roughly the same cost as ignoring errors. Applications with short checkpoint intervals and high write-through penalties will actually observe a performance decrease. Clearly write-through mode provides fault tolerance, but the associated performance penalty provides motivation to seek other solutions.

To test the effectiveness of application-assisted error recovery we have implemented a "rally and recover" error handler that uses fast checkpointing to recover from a parity error. In this strategy a backup copy of the positions and velocities of the atoms is made in

[1]This simulation was performed with a simpler version of the EAM potentials that contained less physics than the EAM potentials implemented currently.

memory every few time steps. The overhead to keep such a copy is small: only a few Mbytes per processor. When a parity error occurs the handler sets an application level flag and instructs the task on which the error occurred to continue calculating even though data have been corrupted. At designated rally points all tasks check the error flag—if the flag is set the current results are discarded and all tasks back up in time by restoring the previously saved positions and velocities. This scheme differs from prior checkpoint based error recovery systems in that the process of checking and communicating error status, as well as saving/restoring checkpoints is under the control of the application rather than the system. This avoids the complications inherent in writing generic recovery code and leverages application specifics to minimize memory requirements and simplify the code needed to check error states and recover from checkpoints.

For ddcMD there is a 3% performance penalty when application-assisted recovery is enabled. This penalty is almost completely due to overhead added by the processor's load pipeline to enable proper error recovery. The cost of the checkpoint operations is negligible. In runs with our error handler enabled the time to recover from a parity error was reduced to the time needed to recompute a small number of time steps (typically a second or less), enabling ddcMD to run continuously at peak performance over long periods. Hence, when the application takes some responsibility for hardware fault recovery it is possible to achieve improved performance in addition to fault tolerance. With ddcMD, we see a 17% improvement (or speedup of 1.2) compared to either no error handling or write-through mode. Paradoxically, the code has achieved a higher overall level of performance by allowing the hardware to make mistakes.

Figure 2 shows the speedup that can be obtained from application-assisted error recovery compared to having the application crash and restart as a function of the crash penalty time (reboot + restart + 1/2 checkpoint interval). Note that a factor of two speedup is a practical limit. Applications with a crash penalty that exceeds 50% of the mean time between failures can limit crash losses using write-though mode. The curve for MTBF=8 represents BG/L. As processor count increases the MTBF will decrease if the failure rate remains the same. The other curves show that benefits of fast error recovery are increased for future machines with shorter failure times due to larger processor counts.

As machines larger than BG/L are constructed it will become even more important for applications to be able to assist the hardware in dealing with errors. Robust fault tolerance at the application level offers improved efficiencies in both run time and memory usage than solutions at the OS or hardware levels. Application level fault tolerance is also easier and more cost-effective to achieve than the construction of no-fault computers. Many codes already contain infrastructure to detect and recover from errors common to numerical simulation such as convergence failures or failures to satisfy error tolerances. Such code can be adapted to serve as hardware fault interrupt handlers. We feel that the ability to run processors in an "unsafe" mode will greatly enhance the effective reliability and overall performance of scientific codes, and pave the way for more aggressive computer design in the next generation of massively-parallel computers.

3. OPTIMIZATION FOR BLUEGENE/L

We have previously reported [35, 36] performance in excess of 100 TFlop/s using ddcMD and the interatomic force law known as Model Generalized Pseudopotential Theory (MGPT) [26]. Although the functional form of MGPT is derived from quantum mechanics through a series of approximations that retains much of the many-body character, it is implemented as a purely classical many-body interatomic potential [25]. Even as an approximation, MGPT requires on the order of a million floating point operations per particle, making it an extraordinarily expensive potential to compute. Given this computational burden, much effort was devoted to optimizing our implementation of the MGPT potential function. Besides a highly tuned compute kernel, a rather complex neighbor table algorithm was implemented. The neighbor algorithm tracked multiple cutoffs and was designed to avoid at all costs the evaluation of redundant force and/or energy terms, as well as terms that would evaluate to zero. By reducing the number of redundant calculations performed we decreased the "performance" of the machine as measured by Flop/s but increased the performance as measured by the most important metric: overall time to solution.

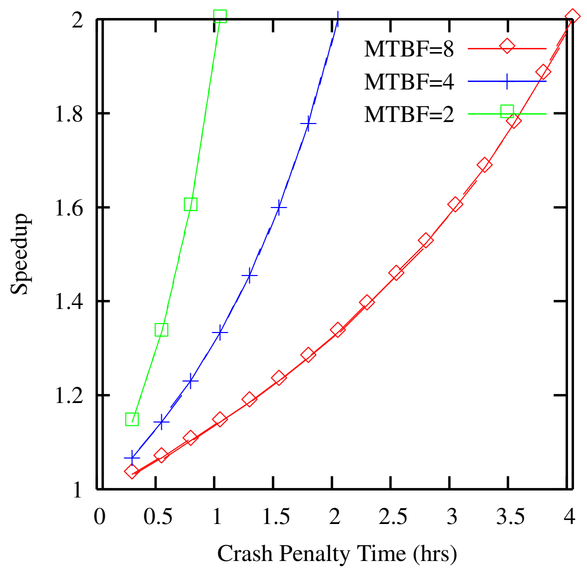

Figure 2: Speedup obtained from application-assisted parity error recovery as a function of crash penalty time for three values of mean time between failures (MTBF): 2, 4, and 8 hours.

With the less expensive EAM potential we found that is no longer optimal to avoid all redundant calculation and communication. The cost to determine and communicate parts of the calculation that are unneeded exceeds the cost of the small number of extra energy and force calculations. Our neighbor list and communication algorithms are now configurable according to the computational demands of the potential to yield the fastest time to solution. As shown in Table 1 even with some redundant calculation allowed, ddcMD still updates atom positions at 3 times the rate of SPaSM for potentials of comparable complexity and cutoff range.

The trade-off between extra calculation and update rate is not always so favorable. Referring again to Table 1 consider the 16 million atom LAMMPS and the 17 million atom MDGRAPE simulations. Both runs are simulating biological systems of roughly the same size using similar potentials. The most computationally demanding part of these simulations is the evaluation of the Coulomb field. MDGRAPE's approach is simply to choose a large cutoff and neglect the contribution of the long-range part of the Coulomb field. LAMMPS on the other hand uses the more efficient and accurate particle-particle-particle mesh method (PPPM) based on Ewald method and FFTs. The PPPM method retains the long-range part of the Coulomb field and does it with good computational ef-

Code	Machine	Cores	Potential	# Atoms	Cutoff (Å)	Updates/sec	Atom-Flop/s/atom	Atoms in cutoff	TFlop/s
LAMMPS[4]	BG/L	65536	LJ	4.0e10		6.86e9	6.28e2	55	4.3
SPaSM[21]	BG/L	131072	LJ	3.2e11	5.85	11.24e9	2.42e3	71	27.2
SPaSM[21]	BG/L	131072	LJ	3.2e11	11.69	2.50e9	1.92e4	565	48.1
SPaSM[12]	BG/L	65536	EAM	6.4e10	4.68	1.65e9	a)	36	a)
ddcMD	BG/L	131072	EAM	2.0e09	4.50	10.12e9	5.37e3	31	54.4
ddcMD[36]	BG/L	131072	MGPT-U	5.2e08	7.24	1.18e8	9.09e5	77	107.6
MDGRAPE[27]	MDGRAPE-3	2304	AMBER	1.4e07	30.00	4.03e7	1.37e6	11,414	55.0
MDGRAPE[14]	MDGRAPE-3	4300	AMBER	1.7e07	44.50	4.14e7	b)4.47e6	37,252	185.0
LAMMPS[4]	Red Storm	512	CHARMM	1.6e07	c)	1.20e7	a)	c)	a)
LAMMPS[4]	Red Storm	10000	CHARMM	3.2e08	c)	2.19e8	a)	c)	a)

Table 1: A collection of large scale MD simulations comparing performance measures. Note a) No Flop information is provided in the reference. Note b) Neither update rates or Flops per atom provide in the reference. Flops per atom was inferred from the Flops per atoms for the 1.4×10^7 atom MDGRAPE run by scaling with $(44.5/30.0)^3$. This number and Flop rate were used to infer the update rate. Note c) LAMMPS has no cutoff for the Coulomb field. The short range part of the interaction is cutoff at 10Å.

ficiency. Despite the enormous advantage in peak Flop/s (850 vs. 41 TFlop/s), the MDGRAPE code does not attain a substantially more rapid time to solution than LAMMPS because of the inefficient treatment of the Coulomb interactions.

3.1 Kernel Optimization

Rather than utilize MGPT in the current study of KH instability, we employ the widely used EAM potentials for metals [5]. These potentials can be characterized as arising from the addition of a multi-body "embedding energy" term to a standard pair potential interaction:

$$E = \sum_i e_i + F(\rho_i)$$
$$\text{where } e_i = \frac{1}{2}\sum_{j \neq i} \phi(r_{ij}) \text{ and } \rho_i = \sum_{j \neq i} f(r_{ij}) \tag{1}$$

The distance between atoms is given by r_{ij}, and the functions f, F and ϕ depend on the type of atoms interacting and their nonlinear forms are specified in the definition of the potential. This potential is computationally inexpensive compared to MGPT, needing only a few thousand floating point operations per particle per evaluation.

There are numerous opportunities to optimize the performance of EAM potentials within ddcMD. In this section we will discuss a few of the most important steps that we have taken as well as further opportunities for performance enhancement that exist.

On the BG/L architecture there are three main issues (other than communication) to consider when optimizing compute-intensive codes such as ddcMD and other molecular dynamics implementations. First, as on all architectures, there is the issue of blocking and cache level. Lower levels of cache can feed the processor with greater bandwidth and with less latency. Second, as is the case on many modern architectures, there is the issue of SIMD code generation. BlueGene/L has a two-way SIMD ISA and a commensurate possibility of a two-fold speed-up for scientific codes. Third, there is the issue of high-precision estimates of operations such as reciprocal and square root. Properly scheduled, these instructions can greatly speed up routines that are dependent upon the calculation of these functions, which are common in molecular dynamics algorithms.

Three routines of interest for optimization were identified: *locator* that finds pairs of interacting atoms, *enden* that computes the 2-body energy and density, and *forstress* that evaluates the forces and stress tensor on the interacting atoms.

The *locator* routine locates all of the atom pairs that are within a prescribed cutoff distance from one another (including periodic boundary conditions). The indices of these atoms are placed in a vectorized list for processing by the other routines. Here, there are three impediments to optimization that had to be overcome. The first is that the atoms are not so local so as to fall into the L1 cache, therefore the routine is currently somewhat bandwidth restricted. The second is that our primary goal in designing the software was to make it portable and easily to debug, therefore the objects used did not always lend themselves to SIMDization, and we were hesitant to use assembly language programming unless performance was simply unacceptable without that level of optimization. Finally, most of our tests for proximity will result in a failure (i.e. most candidate atom pairs are not within the cutoff distance)—many tests are performed and very little action is taken when a test is positive. This high miss rate leads to the evaluation of the branch being a possible bottleneck.

Without rewriting the surrounding framework, the best course of action here was to turn up the compiler optimization level and to unroll the evaluation loop by a factor of two. Since the evaluation is not currently SIMDized and the data are not coming from the L1 cache, this change alone largely put the bottleneck back on the memory subsystem. This change led to a speedup of almost exactly two times while further levels of unrolling did not help performance. Future optimizations are likely to include changing the structures used so that the SIMD nature of the machine can be better exploited and, potentially, interleaving data structures.

The issues involved in the *enden* and *forstress* routines are largely the same, so they will be discussed together here and the differences delineated below. Both *enden* and *forstress* use the list of interacting particles constructed by *locator* and both apply (different) functions to the particles of interest, where the function is dictated by both the routine and the types of the particles under consideration.

The functions $\phi(r_{ij})$ and $f(r_{ij})$ and their derivatives in the EAM potential [23] are given analytically by a combination of exponentials and non-integral powers, however, for these calculations we replace these representations by 32 term polynomial expansions that recovers the original form in the domain of interest. The evaluations of $\phi(r_{ij})$ and $f(r_{ij})$ (and derivatives) in polynomial form are nicely vectorizable (and SIMDizable) and run at very nearly the peak rate of the machine. In fact, the bandwidth demand is such that the polynomial evaluation can run at very nearly 100%

of the architecture's theoretical limit once it is properly unrolled and scheduled. In order to optimize these routines, the most crucial issue was getting the system to produce the desired SIMD instructions. This was done through the use of so called "built-ins," or "intrinsics," that generate the desired (assembly) code without placing the burden of details such as register allocation and instruction scheduling on the programmer. For the computation of polynomial functions this approach worked admirably. However, the routines themselves run at approximately 40% the peak rate of the machine due to a number of now-understood issues that we are attempting to mitigate.

One such issue is directly related to the use of intrinsics, rather than hand-coded assembly instructions. While intrinsics are more readable than assembly instructions and considerably easier to intermix with standard C code, they can lead to scheduling and instruction selection shortfalls that a programmer who understands the architecture might not make. For example, the intrinsics fail to inform the compiler that one piece of data is in the L1 cache (and, therefore, does not need to be prefetched) while another is likely to be in main memory and therefore could greatly benefit from prefetching. Instead, everything is scheduled as if it resides in the L1 cache. Further, the load-and-update instruction, which would likely prove beneficial to the performance of this loop is not used in the generated code.

Anther impediment to performance is both architectural and algorithmic in nature. The current construction of the interaction list produces some atoms with a small neighbor count. Because the register use-to-use time is at least five cycles in the BG/L architecture, we have to have at least 5 interaction evaluations evaluated in the same loop in order for the system to proceed at near peak. More simply said, the evaluation of a single, non-interleaved, particle interaction will take as long as five interleaved reactions. We are currently pursuing strategies to optimize the organization of the neighbor list.

As was mentioned above, *forstress* has additional code for computing the stress tensor. Currently, this is not SIMDizable, but we are considering a slight restructuring of the code and storage methods that should facilitate SIMDization on these, admittedly small, sections of the algorithm.

3.2 Particle-Based Domain Decomposition

We retain the innovative domain decomposition algorithm implemented in ddcMD [36], which was central to the outstanding performance achieved by the code using the expensive MGPT potentials. Our particle-based decomposition strategy allows the processors to calculate potential terms between atoms on arbitrarily separated domains. Domains do not need to be adjacent, they can be arbitrarily shaped and may even overlap. A domain is defined only by the position of its center and the collection of particles that it "owns." This flexibility has a number of advantages. The typical strategy used within ddcMD is initially to assign each particle to the closest domain center, creating a set of domains that approximates a Voronoi tessellation. The choice of the domain centers will control the shape of this tessellation and hence the surface to volume ratio for the domain. It is this ratio for a given decomposition and choice of potential that set the balance of communication to computation. The commonly-used rectilinear domain decomposition employed by many parallel codes is clearly not optimal from this perspective. The best surface to volume ratio in a homogeneous system can be achieved if domain centers form a *bcc*, *fcc*, or *hcp* lattice, which are common high density packing arrangements of atomic crystals.

Even though the best surface to volume ratio would optimize communication cost, load imbalances that may arise (e.g., due to a non-uniform spatial distribution of particles around voids or cracks) requires more flexibility. The domain centers in ddcMD are not required to form a lattice—the application is free to choose any set of domain centers. The flexible domain strategy of ddcMD allows for the migration of the particles between domains by shifting the domain centers. As any change in their positions affects both load balance and the overall ratio of computation to communication, shifting domain centers is a convenient way to optimize the overall efficiency of the simulation. Given the appropriate metric (such as overall time spent in MPI barriers) the domains could be shifted "on-the-fly" in order to maximize efficiency. Currently the domaining is steered dynamically by the user, but it could be implemented automatically within ddcMD.

4. PERFORMANCE

In a perfect world all computing hardware would provide an accurate count of floating point operations and evaluation of performance. However, gathering such information on BG/L using hardware alone is difficult for two reasons: First, not all floating point operations (Flops) are counted, and second, of those operation that can be counted it is not possible to count them all simultaneously.

The 440D architecture of BG/L offers a rich set of floating point operations to manage the two floating point units on the core. There are two broad classes of fp instructions, SISD and SIMD. SISD operations only use the first floating point unit (fp0). The second float point unit (fp1) cannot be accessed independently of fp0 and requires the use of the SIMD instructions. The hardware floating point counter can count four different types of events, but only one at a time. The first type of event is SISD add and subtract (type 0); the second is SISD multiply and divide (type 1); the third is the so-called fused multiply-add and its variants (type 2). These first three event types cover almost all the SISD fp instructions. The fourth and final type of event is the SIMD fused multiply-add operation and its variants. Other SIMD floating point operations are not counted, including all of the non-fused SIMD operation; for example, simple adds and multiplies, as well as float point negation (multiply by -1.0) are not counted.

The inability to count all floating point operations can result in an underestimation of performance, especially for highly optimized code that exploits the second floating point unit (fp1). Such is the case for the kernels used to evaluate the EAM potential in ddcMD. These kernels have a significant number of fused and non-fused SIMD operations. It is possible to count instructions in a basic block of the kernel by looking at the assembler listings. With knowledge of the iteration count for these blocks, an estimate for the missing Flops can be made. Fortunately, for the floating point intensive kernels of the EAM potential the ratio between counted and non-counted SIMD instruction is fixed for each kernel, and that ratio can be found by examining the assembler listing. Simple scaling of the hardware SIMD fp count (event type 3) determines this correction. Overall, we find that a few percent of the Flops are not counted by the hardware counters

Since each task has only one counter it can count only one type of event at a time. To count all events on all tasks for a given calculation the calculation would need to be run four times, (once for each fp event/group) and the total floating point count accumulated at the end. Although this strategy may be feasible for small benchmark runs it is impractical for large science runs. Another approach is to statistically sample the various events by having different tasks count different events. If we divide the four events types between four equally sized sets of tasks, then in principle a statistical measure of the Flop count can be made. This approach is accurate when each task has the same computational profile, however, for

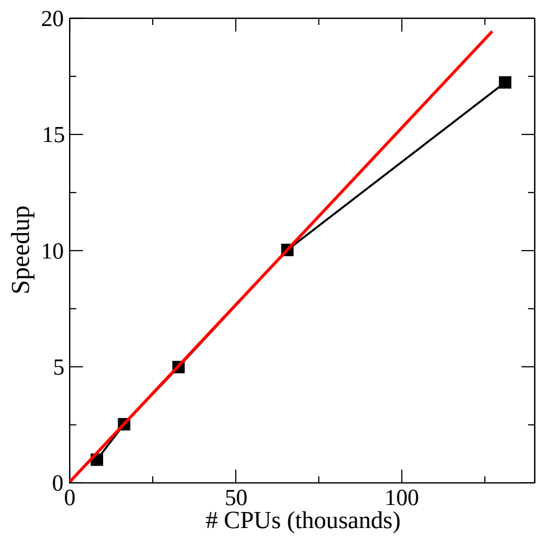

Figure 3: Strong scaling results for ddcMD on BlueGene/L. The red line represents perfect scaling.

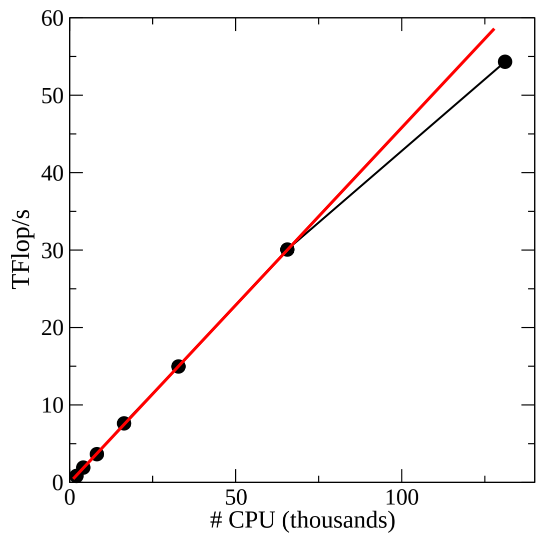

Figure 4: Weak scaling results for ddcMD on BlueGene/L. The red line represents perfect scaling.

our runs this is not the case. On benchmark runs we see that the sampling approach underestimates the Flop count by a few percent when compared to a count made using four separated runs.

4.1 Scaling Benchmarks

All the benchmarks run were run on a thin slab of Cu atoms with 3d periodic boundary conditions. The geometry of the box was 23.5Å $\times$ 1.41s $\times$ s, where s = 3663.24, 5180.86, 7326.47, 10361.73, 14652.95, 20723.45, 29305.89 Å, for 2k, 4k, 8k, 16k, 64k, and 128k tasks respectively (k=1024). The performance data were collected using the sampling procedure previously described.

The rows at 134 $\times$ 10^6 and 536 $\times$ 10^6 atom represent strong scaling studies. Down to about 8000 atoms per task the strong scaling is very good (see Fig. 3), in fact, it shows super linear scaling. The falloff in performance for 128k tasks is to be expected given

# Atoms	Number of Tasks (k=1024)						
($\times 10^6$)	2k	4k	8k	16k	32k	64k	128k
33.6	0.8						21.1
67.0		1.9					30.8
134	0.7	1.6	3.7	7.6	14.3	26.1	38.9
268				7.6			45.6
536			3.0	7.5	15.0	29.8	52.0
1073						30.1	52.5
2146							54.3

Table 2: Performance (TFlop/s) for different system sizes and task count.

the small number of particles per task. The atom counts per task in these cases are 1022 and 2044 respectively for the 134 $\times$ 10^6 and 536 $\times$ 10^6 atom runs. The falloff at a small number of tasks (large number of particles per task) is likely due to increase in cache misses caused by the increased memory footprint.

The complete diagonal of Table 2 represents a weak-scaling study (see also Fig. 4). The data show almost perfect scaling with the exception of the 2k and 128k points. The falloff at large number of tasks is in general to be expected, however, the observed drop in Flop rate at 128k tasks is somewhat larger than anticipated. The falloff at 2k is surprising and we are currently investigating.

ddcMD demonstrates exemplary scaling across the entire machine. We achieve our highest performance of 54.3 TFlop/s using 131,072 processors on a 2146 $\times$ 10^6 sample.

4.2 Science Run Performance

In this section we discuss the performance of the current optimization of ddcMD during the initial stages of a simulation modeling the formation and growth of Kelvin-Helmholtz instabilities. The simulation contains an equal number of Cu and Al atoms for a total of 2.03 billion atoms. In order to achieve the length scales needed for growth of this particular hydrodynamic instability we employ a quasi-2D simulation geometry: 2 nm $\times$ 5 μm $\times$ 2.5 μm. Initially, the system consists of molten Cu and Al separated by an planar interface perpendicular to the z-axis at 1.093 μm. Periodic boundary conditions are used in the x- and y-directions with a 1D confining potential in the z-direction. As of press time, this run has completed the first 150,000 out of a million expected total steps with more extensive science runs to be conducted in the fall.

Because Cu and Al have different number densities the multi-species problem has a spatially inhomogeneous computational load; therefore, particular attention must be paid to load-leveling to fully optimize the simulation. A spatially uniform domain decomposition would suffer a severe load imbalance since the Cu domains would contain more atoms than the Al domains. We have addressed this imbalance by choosing a non-uniform domain decomposition that partitions space based on local pair density (closely related to atomic number density). In practice the optimized non-uniform decomposition achieved a 28% performance increase compared to a uniform domain decomposition.

During the 150,000 time steps of the science run completed to date we successfully caught and recovered from two parity errors. An additional four parity errors were caught during testing and two were recovered successfully. One recovery failure occurred because the parity error corrupted data such that one of ddcMD's internal consistency checks failed and caused a program exit. To avoid such failures in the future we will modify the consistency checks to check the parity flag and attempt recovery when it is

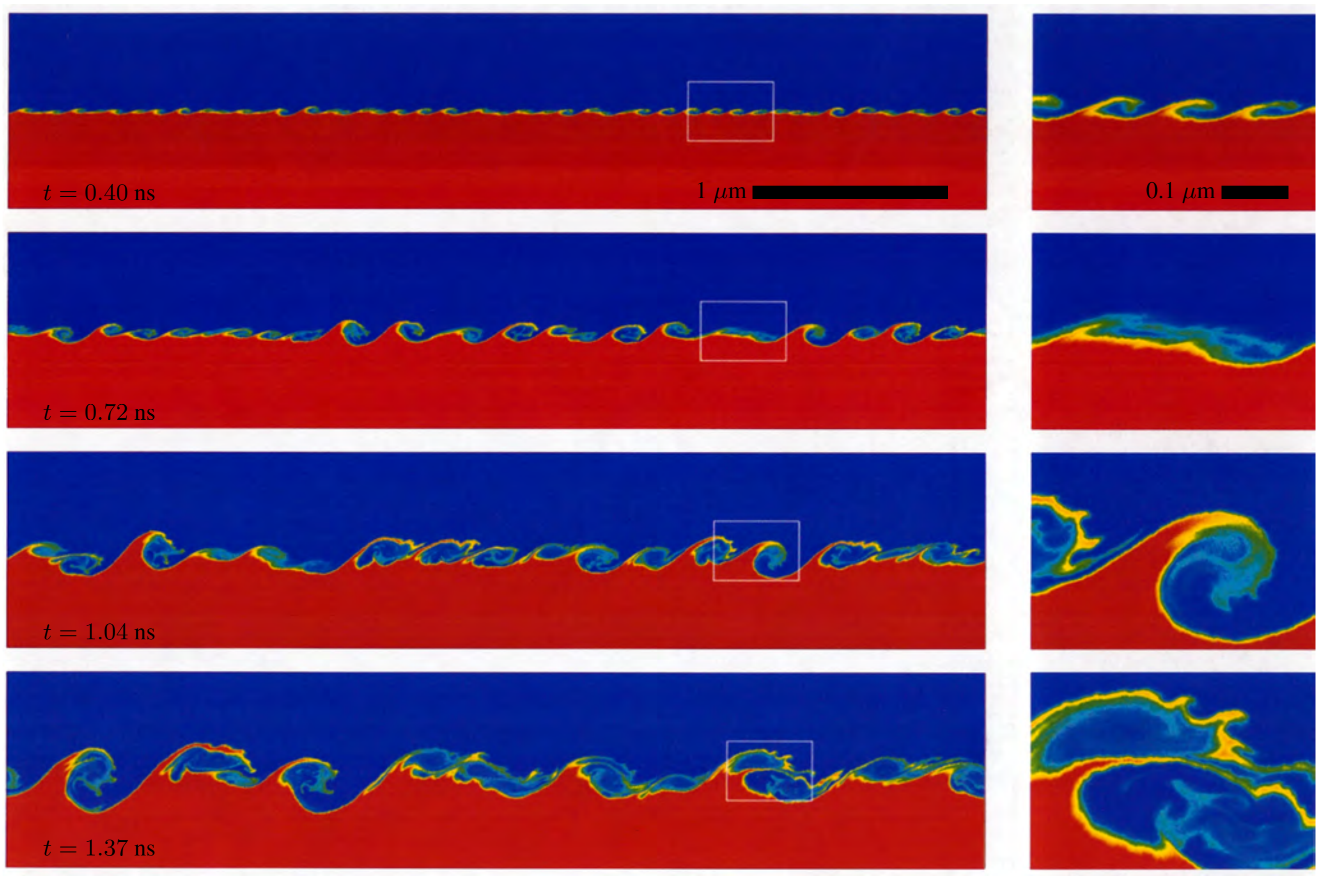

Figure 5: Evolution of Kelvin-Helmholtz instability modeled using molecular dynamics. The color indicates the local density, with red the density of copper and blue the density of Al* (see text). The fluid flows to the right at the top, and to the left at the bottom of each panel. The frames to the right enlarge the outlined rectangular region in the corresponding panel to the left.

set. The cause of the remaining recovery failure is not understood at present. From this limited statistical sample, we estimate that application-assisted error recovery extends the mean time between failures from 8 to at least 48 hours. Future improvements are possible as we enhance our recovery strategy.

We measure the performance of ddcMD using the hardware counters provided by the kernel to sample the four classes of floating point operations in a single run, as described above. The counters tallied the following events over a 27,000 step segment: 2.698×10^{16} type 0, 1.348×10^{16} type 1, 3.007×10^{16} type 2 and 4.704×10^{16} type 3 events in 5407 seconds. Using the appropriate weights for each type, we calculate a total of 2.887×10^{17} floating point operations, for an aggregate performance of 53.4 TFlop/s. As mentioned above the counters do not count all the floating point operations; an analysis of the assembly listing reveals for every 62 events counted by the type 3 counter we miss one floating point operation of weight one and three of weight two. Applying this correction (type 3 count $\times$ 7/62) we calculate our performance to be 54.4 TFlop/s. This simulation is ongoing, and we anticipate presenting results at SC07 that will highlight improved science, stability, and performance.

5. SIMULATION RESULTS

The science results presented in the this section are from an exploratory study of the feasibility of simulating the onset and growth of KH instabilities on the micron scale with atomic resolution. These simulations were successful to this end and have provided a wealth of physical science insight. Although the floating point performance was modest, the computational length of the simulations (2.8 CPU millennia) is unparalleled.

The Kelvin-Helmholtz (KH) instability produces wave patterns at the interface between two fluid layers that are experiencing shear flow. Although the development of the KH instability and the transition from smooth to turbulent flow have been extensively studied, the trend towards smaller and smaller length scales in both experiments and continuum modeling raises questions concerning the applicability of the hydrodynamic approximation as atomic lengths are approached.

The understanding of how matter transitions from an effectively continuous medium at macroscopic length scales to a discrete atomistic medium at the nanoscale is the subject of vigorous academic investigation. Many scientists are pursuing the fundamental question of how far down in size continuum laws apply [32]. This question is not just the subject of arcane academic debate: applications such as the National Ignition Facility are producing gradients on extremely short time and spatial scales while nanotechnologists are studying flow through carbon nanotubes and other systems in which the fluids are but a few atoms thick [16]. Can these phenomena be understood using continuum analysis? What would be the signature that these fluids are discrete? No one has a good answer.

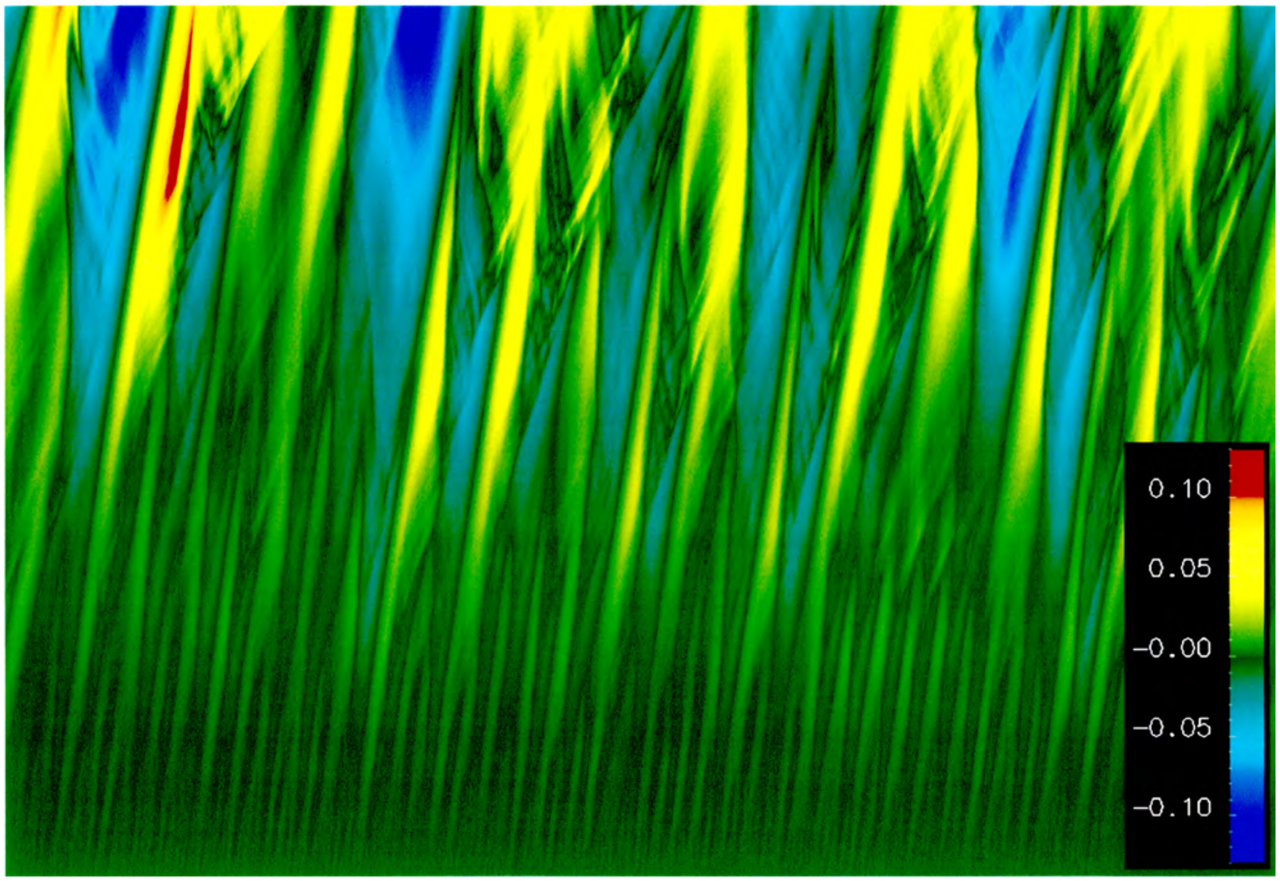

Figure 6: This y-t diagram shows the time evolution of the mass distribution in y, colored such that red indicates a high mass (more copper) and the blue a lower mass (more aluminum). This roughly corresponds to the location of the interface as shown in Fig. 5, with red (blue) indicating a higher (lower) interface. The large swaths of red and yellow indicate large Kelvin-Helmholtz waves, whereas the smaller streaks of yellow at a lower angle indicate material such as in the ligaments being swept from the KH waves. The increase in structure size with time is quite evident as the short wavelength modes at earlier times (bottom of figure) evolve into longer wavelength modes.

In continuum hydrodynamics simulations the macroscopic properties of matter such as pressure, temperature, and flow fields are defined as the collective properties of a sufficiently large ensemble of atoms. Continuum equations such as the Navier-Stokes equation are derived from conservation laws assuming that field gradients are small and that material properties change slowly compared to atomic length and time scales. Although the Navier-Stokes equation provides a very powerful description of the continuum limit it is predicated on an asymptotic analysis. If the gradients become too large, there is no way to fix Navier-Stokes by adding higher order terms in the gradients to construct a convergent series—the math does not work that way. The situation is even more complicated in fluids due to their chaotic nature.

In contrast to hydrodynamics, molecular dynamics (MD) utilizes no direct continuum-level input. Instead of a continuum constitutive law, MD is based on an interatomic force law. Properties such as the equation of state (density as a function of temperature and pressure), transport coefficients such as the diffusivity and the viscosity, and interfacial properties such as the surface tension arise naturally from these underlying atomic forces. Additionally, with a suitable time step MD is unconditionally stable, as the system is evolved using an explicit time integrator. MD is fully resolved by nature—there is no mesh size to adjust, and no gradient is too steep. Inter-diffusion at interfaces is physical, not a numerical artifact. Numerically, MD simulation is the gold standard. Unlike a hydrodynamic simulation, where the challenge is to add sufficient degrees of freedom to obtain a converged result, the challenge for MD has always been to overcome the constraints on system size and simulation length imposed by computational resources.

We have used the ddcMD code to simulate the development of the Kelvin-Helmholtz instability at an interface between two different kinds of molten metal flowing in opposite directions, as shown in Fig. 5. The initial atomic configuration was constructed in a simulation box 2 nm $\times$ 5 μm $\times$ 2.9 μm in size containing two types of atoms with a total of $2{\times}10^9$ atoms (1 billion of each type). Periodic boundary conditions were used in the x- and y-directions; i.e., the thin direction into the page and the horizontal flow direction in Fig. 5. The third dimension, z, was not periodic: a static potential based on the atomic pair potential was used to confine the atoms. The use of the confining potential permitted the system to have a single interface between the two kinds of fluid, gaining a factor of two in computational efficiency over a 2-interface fully periodic system. The initial velocity of each atom was selected at random from a Boltzmann thermal distribution to give a temper-

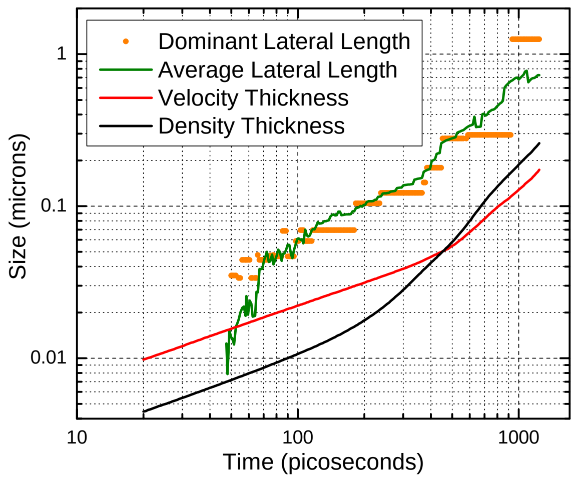

Figure 7: Kelvin-Helmholtz length scales as a function of time. Three relevant length scales of the KH development are plotted, one of which is plotted in two ways. The mass and velocity thicknesses are plotted as a function of time (as defined in the text), indicating the vertical thickness of the interface as plotted in the snapshots in Fig. 5. Also the dominant length scale along the interface is plotted in two ways: first, the line segments indicate the size of the dominant wavelength in the Fourier transform of the mass profile in y; second, the solid curve represents the weighted average of the sizes of the 10 largest peaks in the Fourier transform of the mass function, providing a more smooth representation of the lateral size growth. After an initial diffusive regime, followed by a transient regime, the system enters into a self-similar growth regime in which the aspect ratio of the vortex structures remains constant.

ature of ∼2000K in the local rest frame of its fluid, and the two fluids were given an initial relative velocity of 2000 m/s. This step-function initial velocity profile sets up a strong shear layer at the flat interface between the two metals, which is known to be unstable against the growth of small perturbations.

The atoms were chosen to simulate molten copper and aluminum. Both interact with forces derived from a classical EAM potential. A common EAM potential for copper was used to simulate the interatomic forces for both types of metals [18, 17], and the masses of the two kinds of atoms were taken to be m_{Cu}=63.546 AMU and m_{Cu}/3, to give a fluid density approximating that of aluminum[2]. We refer to the two fluids as Cu and Al*.

Newton's equations ($F = ma$) based on the EAM force law were integrated in time using an explicit symplectic time integrator with a time step of 2×10^{-15} s. The system was evolved for over 680,000 time steps, giving a simulated period of over 1.3 nanoseconds. The volume and the number of atoms were held constant. No thermostat or velocistat was used; the flow velocity was maintained by inertia, remaining at ∼2000 m/s throughout the simulation.

The effect of the KH instability on the interface is quite pronounced, as shown in Fig. 5. Initially, atomic-level inter-diffusion leads to a broadening of the interface. This inter-diffusion layer maintains a flat interface on the average, but atomic-level thermal fluctuations perturb the interface and trigger the growth of wave structures. These structures grow in amplitude and eventually crest to form micron-scale vortices. As the KH instability grows vertically, the characteristic wavelength of the waves and vortices grows in the direction of the flow as well. Initially short wavelength modes grow, but ultimately the larger structures grow at the expense of the small ones. This ripening may be seen in Fig. 6, in which the evolution of the interface height with time is plotted. A similar plot has been used to analyze shock-accelerated interfaces [15]. The initial short wavelength structure evolves into the much larger vortex structures as evident from the broad bands, with fine structure arising from the ligaments and other material transport processes in the vortices. We have analyzed the growth effects quantitatively by calculating the interface width, both in terms of the material profile and the velocity profile. In particular, generalizations of the momentum thickness formula were used,

$$T_u = 6 \int \frac{u - u_0}{u_1 - u_0} \frac{u_1 - u}{u_1 - u_0} dz, \qquad (2)$$

where $u(z)$ is the profile of a function averaged over x and y, with far field values u_0 and u_1. We have also calculated the principal feature size of the interface both as the size associated with the dominant peak in the Fourier transform of the mass function $m(y)$, and as the weighted average of the sizes associated with the 10 largest peaks. These quantities are plotted in Fig. 7. Early in time, the interface grows in a regime associated with momentum diffusion, in which the interface widths (both the velocity and density thicknesses) grow as the square root of time. The interface development then passes through a transient regime until at late times self-similar growth appears to set in, where the interface widths and the dominant structure sizes grow with the same power-law exponent so that the aspect ratio of the vortices is maintained at approximately two to one. The spectrum of fluctuations that initiates this growth is at the atomistic level, but the momentum diffusion and vortex regimes are characteristic of continuum hydrodynamic behavior. These simulations have therefore spanned physics at the atomic level to continuum hydrodynamic length scales.

6. CONCLUSIONS

These simulations have opened the door to many possibilities for studying the various physical processes associated with the Kelvin-Helmholtz instability at length scales spanning atomic to continuum hydrodynamic levels. The ddcMD code, with its particle-based domain decomposition and highly refined kernel, has provided the performance needed to make efficient use of BG/L. With the trapping of hardware errors demonstrated here we show that stability on modern massively parallel machines can be extended to unprecedented levels without a significant loss of performance.

This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

7. REFERENCES

[1] M. P. Allen and D. J. Tildesley. *Computer Simulation of Liquids*. Clarendon Press, Oxford, 1987.

[2] S. Chandrasekhar. *Hydrodynamic and Hydromagnetic Stability*. Oxford Univ. Press, Oxford, 1961.

[3] M. P. Ciamarra, A. Coniglio, and M. Micodemi. Shear instabilities in granular mixtures. *Phys. Rev. Lett.*, 94:188001, 2005.

[2]In our ongoing simulation we employ a more realistic Cu-Al alloy potential[23]

[4] P. Crozier, F. Reid, and C.Vaughn. Lammps benchmarks, http://lammps.sandia.gov/bench.html.

[5] M. S. Daw and M. I. Baskes. Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals. *Phys. Rev. B*, 29(12):6443–6453, 1984.

[6] P. G. Drazin and W. H. Reid. *Hydrodynamic Stability*. Cambridge Univ. Press, Cambridge, 1981.

[7] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408, 2002.

[8] C. Engelmann and A. Geist. Super-scalable algorithms for computing on 100,000 processors. In *Computational Science - ICCS 2005, PT 1, Proceedings*, volume 3514 of *Lecture Notes in Computer Science*, pages 313–321, 2005.

[9] O. B. Fringer and R. L. Street. The dynamics of breaking progressive interfacial waves. *J. Fluid Mech.*, 494:319–353, 2003.

[10] D. C. Fritts, C. Bizon, J. A. Werne, and C. K. Meyer. Layering accompanying turbulence generation due to shear instability and gravity-wave breaking. *J. Geophys. Res.*, 108:20:1–13, 2003.

[11] D. C. Fritts, T. L. Palmer, O. Andreassen, and I. Lie. Evolution and breakdown of kelvin-helmholtz billows in stratified compressible flows:1. comparison of two- and three-dimensional flows. *J. Atmos. Sci.*, 53:3173–3191, 1996.

[12] T. C. Germann, K. Kadau, and P. S. Lomdahl. 25 tflop/s multibillion-atom molecular dynamics simulations and visualization/analysis on bluegene/l. *Proc. Supercomputing 2005, Seattle, 2005, on CD-ROM*, http://sc05.supercomputing.org/schedule/pdf/pap122.pdf.

[13] M. Gomaa, C. Scarbrough, T. N. Vijaykumar, and I. Pomeranz. Transient-fault recovery for chip multiprocessors. In *ISCA '03: Proceedings of the 30th annual international symposium on Computer architecture*, pages 98–109, New York, NY, USA, 2003. ACM Press.

[14] Grape newletter vol.8 (dec 16 2006), http://grape.mtk.nao.ac.jp/grape/newsletter/061216.html.

[15] J. F. Hawley and N. J. Zabusky. Vortex paradigm for shock-accelerated density-stratified interfaces. *Phys. Rev. Lett.*, 63(12):1241–1244, 1989.

[16] J. K. Holt, H. G. Park, Y. Wang, M. Stadermann, A. B. Artyukhin, C. P. Grigoropoulos, A. Noy, and O. Bakajin. Fast Mass Transport Through Sub-2-Nanometer Carbon Nanotubes. *Science*, 312(5776):1034–1037, 2006.

[17] R. A. Johnson. Alloy models with the embedded-atom method. *Phys. Rev. B*, 39:12554, 1989.

[18] R. A. Johnson and D. J. Oh. Analytic embedded atom method model for bcc metals. *J. Mater. Res.*, 4:1195, 1989.

[19] V. Junk, F. Heitsch, and T. Naab. The kelvin-helmholtz instability in smoothed particle hydrodynamics. *Proc. Int. Astro. Union*, 2:210–210, 2007.

[20] K. Kadau, T. C. Germann, N. G. Hadjiconstantinou, P. S. Lomdahl, G. Dimonte, B. L. Holian, and B. J. Alder. Nanohydrodynamics simulations: An atomistic view of the rayleigh-taylor instability. *Proc. Natl. Acad. Sci. USA*, 101:5851–5855, 2004.

[21] K. Kadau, T. C. Germann, and P. S. Lamdahl. Molecular dynamics comes of age: 320 billion atom simulation on bluegene/l. *Int. J. Mod. Phys. C*, 17:1755, 2006.

[22] H. Lamb. *Hydrodynamics*. Cambridge Univ. Press, Cambridge, 6th edition, 1932.

[23] D. R. Mason, R. E. Rudd, and A. P. Sutton. Stochastic kinetic monte carlo algorithms for long-range hamiltonians. *Computer Physics Comm.*, 160:140–157, 2004.

[24] J. J. Monaghan. Flaws in the modern laplacian theory. *Earth, Moon and Planets*, 71:73–84, 1995.

[25] J. A. Moriarty. Analytic representation of multi-ion interatomic potentials in transition metals. *Phys. Rev. B*, 42:1609–1628, 1990.

[26] J. A. Moriarty. Angular forces and melting in bcc transition metals: A case study of molybdenum. *Phys. Rev. B*, 49:12431–12445, 1994.

[27] T. Narumi, Y. Ohno, N. Okimoto, T. Koishi, A. Suenaga, N. Futatsugi, R. Yanai, R.Himeno, S. Fujikawa, M. Taiji, and M. Ikei. A 55 tflops simulation of amyloid-forming peptides from yeast prion sup35 with the special-purpose computer system mdgrape-3. *Proc. Supercomputing 2006, Tampa, 2006, on CD-ROM*, http://sc06.supercomputing.org/schedule/pdf/gb106.pdf.

[28] T. L. Palmer, D. C. Fritts, and O. Andreassen. Evolution and breakdown of kelvin-helmholtz billows in stratified compressible flows:2. instability structure, evolution, and energetics. *J. Atmos. Sci.*, 53:3192–3212, 1996.

[29] D. C. Rapaport. *The Art of Molecular Dynamics Simulation*. Cambridge Univ. Press, Cambridge, 1995.

[30] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. I. August. Swift: Software implemented fault tolerance. In *CGO '05: Proceedings of the international symposium on Code generation and optimization*, pages 243–254, Washington, DC, USA, 2005. IEEE Computer Society.

[31] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, D. I. August, and S. S. Mukherjee. Design and evaluation of hybrid fault-detection systems. *SIGARCH Comput. Archit. News*, 33(2):148–159, 2005.

[32] R. E. Rudd and J. Q. Broughton. Coupling of length scales in solid state systems. *Phys. Stat. Sol.*, 217:251–291, 2000.

[33] SETI@HOME project: http://setiathome.ssl.berkeley.edu.

[34] Image © 1999 Beverly Shannon. From Clouds over Mount Shasta: http://www.siskiyous.edu/shasta/env/clouds.

[35] F. H. Streitz, J. N. Glosli, and M. V. Patel. Beyond finite-size scaling in solidification simulations. *Phys. Rev. Lett.*, 96:225701, 2006.

[36] F. H. Streitz, J. N. Glosli, and M. V. Patel. Simulating solidification in metals at high pressure: The drive to petascale computing. *J. Phys.: Conf. Ser.*, 46:254–267, 2006.

[37] R. I. Sykes and W. S. Lewellen. A numerical study of breaking of kelvin-helmholtz billows using a reynolds-stress turbulence closure model. *J. Atmos. Sci.*, 39:1506–1520, 1982.

[38] TOP500 Supercomputer Sites: http://www.top500.org/.

[39] J.-Y. Yang and J.-W. Chang. Rarefied flow instability simulation using model boltzmann equations. *Proc. Fluid Dynamics Conf. 28th, Snowmass Village, CO*, June 29–July 2, 1997.

[40] Y. Yeh. Triple-triple redundant 777 primary flight computer. In *Proceedings of the 1996 IEEE Aerospace Applications Conference*, volume 1, pages 293–307, 1996.

[41] R. Zhang, X. He, G. Doolen, and S. Chen. Surface tension effect on two-dimensional two-phase kelvin-helmholtz instabilities. *Adv. Water Res.*, 24:461–478, 2001.