Tracing Communications and Computational Workload in LJS (Lennard-Jones with Spatial Decomposition)

Maciej Brodowicz

CACR, Caltech
September 2003

1. Overview

LJS (**L**ennard-**J**ones with **S**patial decomposition) is a molecular dynamics application developed by Steve Plimpton at Sandia National Laboratories [1]. It performs thermodynamic simulations of a system containing fixed large number (millions) of atoms or molecules confined within a regular, three-dimensional domain. Since the simulations model interactions on atomic scale, the computations carried out in a single timestep (iteration) correspond to femtoseconds of the real time. Hence, a meaningful simulation of the evolution of the system's state typically requires a large number (thousands and more) of timesteps.

The particles in LJS are represented as material points subjected to forces resulting from interactions with other particles. While the general case involves N-body solvers, LJS implements only pair-wise material point interactions using derivative of Lennard-Jones potential energy for each particle pair to evaluate the acting forces. The velocities and positions of particles are updated by integrating Newton's equations (classical molecular dynamics). The interaction range depends on the modeled problem type; LJS focuses on short-range forces, implementing a cutoff distance r_c outside which the interactions are ignored. The computational complexity of $O(N^2)$, characteristic for systems with long-range interactions, is therefore substantially alleviated.

LJS deploys spatial decomposition of the domain volume to distribute the computations across the available processors on a parallel computer. The decomposition process uniformly divides parallelepiped containing all particles into volumes equal in size and as close in shape to a cube as possible, assigning each of such formed cells to a CPU. The correctness of computations requires the positions of some particles (depending on the value of r_c) residing in the neighboring cells to be known to the local process. This information is exchanged in every timestep via explicit communication with the neighbor nodes in all three dimensions (for details see [2]). LJS also takes the advantage of the third Newton's law to calculate the force only once per particle pair; if the involved particles belong to cells located on different processors, the results are forwarded to the other node in a "reverse communication" phase.

Besides communications occurring in every iteration, additional messages are sent once every preset number of timesteps. Their purpose is to adjust cell assignments of particles due to their movement. To minimize the overhead of the construction of particle neighbor lists, LJS replaces r_c with extended cutoff radius r_s ($r_s > r_c$), which accounts for possible particle movement before any list updates need to be carried out. Due to a relatively small impact of that phase on the overall behavior of the application, we ignored it in our analysis.

2. Configuration

The runtime parameters of the simulation along with their values are listed in the table below:

Name	**Value**	**Description**
Physics		
Dt	0.00442	Timestep size in reduced units
T_0	1.444	Initial temperature in reduced units
ρ	0.8442	Density in reduced units
r_c	2.5	Cutoff distance in reduced units
r_s	2.8	Extended cutoff distance in reduced units
Problem definition and execution control		
n_x, n_y, n_z	50, 50, 50 (per CPU)	Dimensions of domain bounding box (integer units)
$alat$	$(4/\rho)^{1/3} \cong 1.68$	Linear scaling factor
T	5	Number of simulation timesteps
n_{neigh}	20	Number of timesteps between re-binning
$nbin_x$, $nbin_y$, $nbin_z$	0.6 n_x, 0.6 n_y, 0.6 n_z	Number of cells per each dimension of the domain

The total number of particles, N, is given as

$$N = 4\, n_x\, n_y\, n_z\,,$$

where n_i are integers (there is a fixed average of four particles per unit cube). The problem is executed on a grid of P processors, such that

$$P = p_x\, p_y\, p_z, \text{ with } p_i = n_i/k_i \text{ where } k_i \text{ are integers.}$$

In our benchmarks $k_i = 50$, hence the problem size was 50x50x50 (or 500,000 particles) on a single, 100x100x50 on four, 100x100x100 on eight and 200x200x200 on 64 processors. Such configurations require approximately 200MB of memory per CPU for all LJS data structures. Note that the cutoff distances are significantly smaller than the linear dimensions of the domain fragment assigned to a single processor ($50 \cdot alat \cong 84$), hence the spatial decomposition algorithm is performing efficiently (time spent in all communication phases is significantly smaller than the total computation time and didn't exceed 15% of the application runtime in our experiments).

LJS initializes its data structures by assigning particle positions on a regular 3-D mesh (thus emulating crystal lattice) and computing velocity vectors to satisfy the initial temperature requirement. The velocities are otherwise random in magnitude and direction. In the next few timesteps of the simulation the particles move from their positions on the grid (the crystal melts). Therefore, to capture the application behavior as close to the average (no imbalances of particle counts between processors), we limited the tracing to the first five iterations.

3. ETF Instrumentation

In order to extract more detailed runtime information, ETF (Extensible Tracing Facility) was used to instrument the application code. The instrumentation's goal was to provide low-level instruction counts executed at the user level, obtain timing information, register parameters used by MPI routines for message passing and mark starting and ending points of important execution phases.

3.1 ETF Counters and Timers

ETF is capable of accessing high-resolution timers and hardware event counters on select platforms. Currently, such support exists for IBM SP2 (Power processors) via PMAPI interface. Using Power architecture in our experiments is convenient, as BG/L processors are based on a modified version of the PowerPC core, which shares significant elements of ISA and hardware features with Power processor line. PMAPI supports up to eight 64-bit event counters, however, they cannot be assigned to counters arbitrarily and that limits the effective number of events monitored concurrently. Another constraint is caused by OS overhead (1200..1500 cycles per readout of a counter set), which may produce skewed results if the counters are accessed too frequently. In our experiments we decided to restrict the monitoring to the execution in user mode only, as the kernel mode offers little insight into application behavior and cannot be properly verified due to lack of the source code. The events of interest included: the number of CPU cycles spent executing the code, the number of instructions completed and the cumulative count of all FPU operations (this required summing the counts of instructions retired by both floating point units of the processor).

3.2 MPI Communication

LJS uses a small subset of MPI-1 calls for message passing. The collective calls (*Barrier*, *Bcast*, *Allreduce*) are invoked only during the setup phase and when computing thermodynamic state of the system (typically at the end of execution). Throughout the simulation, the bulk of data is transferred by point-to-point calls (blocking *Send* and non-blocking *Irecv*, which enable overlapping of bi-directional transmissions). For the parameter set listed in section 2, the messages originating from each node are emitted to its six nearest neighbor (in 3-D grid) nodes only. Due to the use of periodic Cartesian communicator, particles migrating outside the domain from boundary cells in any dimension, appear in the opposite boundary cell in that dimension.

To simplify the trace analysis, ETF was configured to register the message sizes and destination nodes of point-to-point communications. Memory reference tracking, which includes message buffer pointers and maps of MPI datatypes in the trace, was disabled.

3.3 LJS Execution Phases

The source code of LJS was augmented with calls injecting markers at the endpoints of the following phases:

- Setup and initialization (procedures: *input, setup_general, setup_memory, setup_comm, setup_neigh, setup_atom, scale_velocity, exchange, borders, neighbor*)
- Iteration of the main loop (*integrate*):
  - Calculation of the new positions of particles
  - Communication: update of the positions of remote particles (*communicate*)
  - Computation of forces (*force_newton*)
  - Reverse communication: propagation of forces (*reverse_comm.*)
  - Calculation of particle velocities
- Final thermodynamics evaluation and printout (*thermo, output*)

4. Tracing Results

4.1 Computational Profile

The computational workload was very consistent from iteration to iteration and across the nodes. This is expected due to uniform initial distribution of particles and symmetric neighborhoods of each cell. The tables below present the counter values collected for the setup, intermediate phases of the fourth timestep of the simulation (which is representative for other iterations as well) and finalization phase for different number of processors.

1 CPU, grid: 50x50x50	Cycles	Instructions	FPU ops
Setup	5052199616	4429953927	1770200815
without MPI	5047941630	4426692460	1770199937
Position computation	10177454	6250758	1500002
Communication	5773665	1296831	337623
Compute force	906525552	646414360	311087829
Reverse communication	3984631	1633667	337586
Velocity computation	24412276	8750806	1500003
Statistics and output	1582670466	1108981709	543052406
without MPI	1582176191	1108745775	543051159

4 CPUs, grid: 100x100x50	Cycles	Instructions	FPU ops
Setup	5221111514	4511174346	1778322208
without MPI	5182718759	4474679853	1778297561
Position computation	14830811	6250758	1500004
Communication	10173647	3452399	247947
without MPI	4789403	1383933	247539
Compute force	919677779	642524200	309253334

Reverse communication	20714843	17943413	403433
without MPI	3452535	1923312	393767
Velocity computation	28379907	8750806	1500002
Statistics and output	1629528274	1127778373	539980682
without MPI	1605565123	1102436748	539957989

8 CPUs, grid: 100x100x100	Cycles	Instructions	FPU ops
Setup	5219709948	4559748718	1778872028
without MPI	5148088364	4486202543	1778823102
Position computation	19109039	6250758	1500012
Communication	12344903	3682710	193669
without MPI	5402195	1179174	192966
Compute force	904651811	642808360	309405641
Reverse communication	69268341	71835364	383606
without MPI	4381723	1660865	337595
Velocity computation	33611634	8750806	1500004
Statistics and output	1611655969	1126834265	540171202
without MPI	1588583802	1102777718	540147789

64 CPUs, grid: 200x200x200	Cycles	Instructions	FPU ops
Setup	8482763222	7700183441	1782447444
without MPI	5187570720	4486216886	1781271594
Position computation	18828594	6250744	1500010
Communication	21939538	14506940	197011
without MPI	5066046	1179028	192948
Compute force	905015895	642808346	309131305
Reverse communication	79774656	72790176	362519
without MPI	4787509	1660719	337607
Velocity computation	33838233	8750792	1500003
Statistics and output	1778038079	1283720429	540689376
without MPI	1590583078	1102771188	540609864

The only significant inconsistencies are variances in cumulative event counts for the communication phases. This is understandable, since the message passing is inherently non-deterministic. For example, messages of identical sizes can be split into different number of packets depending on the transient condition of the interconnect network and hence the overhead of message fragmentation and reassembly may not be identical. Note that even though the network traversal time should be excluded from timings in user mode, the actual behavior is strongly implementation dependent; if the MPI library uses busy waiting to poll for incoming messages, this fact will be reflected in counts. For that reason we included additional entries for setup, communication and output phases showing operation counts without the overhead of MPI wrapper execution. Even for the cumulative counts, however, the global trend of increasing the overhead with the problem size is sustained.

LJS deploys a “leapfrog” integrator, whose operation is expressed as (only position computation shown; velocity calculation is identical in complexity with properly adjusted *dt*):

$$x_i(t+1) = x_i(t)+v_i(t+1/2)\ dt,$$ for dimension $i = 1, 2, 3$ in iteration t.

This is in nearly perfect agreement with the FPU counts: 500,000 particles per CPU with 3 dimensional components yield 1.5 million operations. The compiler takes advantage of the fact that Power ISA includes a multiply-add instruction; otherwise the FPU counts would be twice as high. The number of cycles spent in velocity calculation phase is higher than that of position integration, since the previous values of velocity vectors need to be preserved for thermodynamic state computations, while the old position vectors are simply overwritten. The copy operation doesn’t use the FPU, hence the additional overhead manifests itself only in increased instruction/cycle counts. Still, by far the most dominant portion of each timestep is devoted to the force computation, thus justifying the presence of reverse communication step.

4.2 Communication Profile

Due to symmetry of the problem decomposition and repeatability of parameters passed to MPI calls, only one-iteration behavior on a single processor was analyzed. The results were collected in the table listing destinations and sizes of messages transmitted from rank 0. The send order in each communication phase is reflected by the row position (entries closer to the top of the table are sent earlier).

Configuration	Comm. phase	Destination rank	Message size (bytes)
4 CPUs	Forward	2	496800
		2	372600
		1	1063152
		1	797352
	Reverse	1	797352
		1	1063152
		2	372600
		2	496800
8 CPUs	Forward	4	480000
		4	360000
		2	513600
		2	385200
		1	549552
		1	412152
	Reverse	1	412152
		1	549552
		2	385200
		2	513600
		4	360000
		4	480000
64 CPUs	Forward	48	480000
		16	360000

		12	513600
		4	385200
		3	549552
		1	412152
	Reverse	3	412152
		1	549552
		12	385200
		4	513600
		48	360000
		16	480000

This scheme is repeated in every timestep; the differences across the nodes are relevant only to destination rank numbers, but they stay fixed throughout the execution for a given sender node. Note that the number of messages is reduced when running on less than 8 processors. This is because 2^3 CPUs is the smallest configuration where the computational domain can be decomposed into at least two partitions along each dimension. A small inefficiency of LJS may be observed when running on less than 64 processors: the messages within the same communication phase are emitted to repeated destinations. When scaling beyond 64 CPUs, message sizes and number of destinations stay fixed as long as the size of local grid on every processor is preserved.

4.3 Algorithm Scaling

To verify the characteristics of program execution for other problem sizes, LJS was traced with reduced grid size of $n_x = n_y = n_z = 100$ on 64 processors. The computational workload parameters (fourth iteration only) and message sizes were collected in the following tables:

64 CPUs, grid: 100x100x100	Cycles	Instructions	FPU ops
Setup	4518993233	4403610464	224582074
without MPI	631111169	567804724	223936578
Compute positions	1130213	781994	187500
Communication	4574366	3009445	52475
without MPI	590020	337116	51639
Compute force	109366444	79949995	38478599
Reverse communication	7547645	6008098	92208
without MPI	771809	465557	90311
Compute velocities	3222944	1094542	187501
Statistics and output	206037876	148078372	67258754
without MPI	193958062	137728315	67242629

Configuration	**Comm. phase**	**Destination rank**	**Message size (bytes)**	**Ratio to msg. size for 200^3 grid**
64 CPUs, 100x100x100 grid	Forward	48	120000	1:4
		16	90000	1:4
		12	136800	1:3.75
		4	102600	1:3.75
		3	155952	1:3.52

		1	116952	1:3.52
	Reverse	3	116952	1:3.52
		1	155952	1:3.52
		12	102600	1:3.75
		4	136800	1:3.75
		48	90000	1:4
		16	120000	1:4

As can be easily seen, the computational workload decreased proportionally to the problem volume (2^3 times, as the problem size in each dimension was halved). The memory allocation for LJS data arrays was 26.5MB per processor, again – roughly 1/8 of that required for 200x200x200 configuration. Message sizes were reduced approximately four times, what agrees well with the assumption of communication volume being proportional to the cell surface area. Note that the ideal ratio of four is observed only for the exchanges along the first dimension; this figure is distorted for transmissions along the second and third dimension due to the fact that the presence of volume characteristic decreases in subsequent data sends.

The spatial decomposition algorithm implemented in LJS behaves consistently over wide range of grid sizes. The anomalies resulting from the cutoff distance r_s being comparable with the physical dimensions of a sub-grid assigned to a single processor arise for relatively small problem sizes, for which the communication overhead nearly always exceeds the cumulative duration of computations. To investigate such a case, the program was configured to run a 20x20x20 problem (on 64 CPUs this yields 5x5x5 grid per processor) with artificially increased values of $r_c = 10$ and $r_s = 11.2$. The communication characteristics are presented below with the "reverse" communication phase omitted for brevity.

Configuration	Destination rank	Message size (bytes)
64 CPUs, 20x20x20 grid	48	12000
	16	12000
	48	4800
	16	3600
	12	44400
	4	44400
	12	17760
	4	13320
	3	164280
	1	164280
	3	65712
	1	49272

Since r_s is longer than the linear dimension of the local sub-domain ($d = 5 \cdot alat \cong 8.4$), the communication must involve not only the immediate neighbors of a processor, but also cells located one more grid "hop" away (because $d < r_s < 2d$). As LJS processes don't communicate with the remote neighbors directly, the data are passed in multiple steps

through the immediate neighbors' buffers. Unlike in the typical scenario described in section 4.2, the ratio of communication volume along the third dimension (the last four entries in the table) to that of the first dimension (the first four entries) is significantly larger due to much larger final volume of data accumulated from neighboring cells within the cutoff distance in all three dimensions compared to that for just one dimension. The memory consumption, 10.7MB per processor, also deviates from the simplistic estimates, most likely due to excessive buffer space required for communication. However, such situations arise rarely in practical short-range problems when executed on machines with sufficient amount of memory per node.

References

[1] Steve Plimpton, "Fast Parallel Algorithms for Short-Range Molecular Dynamics", Journal of Computational Physics 117, 1-19 (1995)

[2] Tom Gottschalk, "Scaling and Complexity: Spatial Decomposition MD", white paper