Performance Analysis of a High Energy Colliding Beam Simulation Code on Four HPC Architectures

Hongzhang Shan[1], Ji Qiang[2], Erich Strohmaier[1], and Kathy Yelick[1]
Computational Research Division1
Accelerator and Fusion Division
Lawrence Berkeley National Laboratory
One Cyclotron Road, Berkeley, CA 94720
{hshan, jqiang, estrohmaier, kayelick@lbl.gov}

Abstract

The high energy colliders are essential to study the inner structure of nuclear and elementary particles. A parallel particle simulation code, BeamBeam3D, has been developed and actively used to model the beam dynamics and to optimize the performance of these colliders. In this paper, we analyzed the performance characteristics of BeamBeam3D on four leading high performance computing architectures, including a massive parallel system, a commodity-based cluster, an advanced vector platform, and a novel architecture focused on low power consumption and high density. We examine how to partition the workload among the processors to effectively use the computing resources, whether these platforms exhibit similar performance bottlenecks and how to address them, whether some platforms perform substantially better than others, and finally, the implications of BeamBeam3D for the design of the next generation supercomputer architectures.

1. Introduction

Beam-beam interaction from electromagnetic force of charged particles is a dominant factor limiting luminosity in modern ring colliders, where two beams rotate counterwise with each other and collide at interaction regions. An accurate modeling of the beam-beam interaction is essential to maximizing the luminosity in high energy accelerator ring colliders. It is also critical for building the next generation colliders such as Large Hadron Collider (LHC); any design errors will be enormously costly, or even fatal, for this "big science" project [5]. However, due to the extreme computational cost required to accurately and self-consistently model the beam-beam interaction as the beams circulate for millions of turns, previous studies are confined to use simplified models. Examples include the "weak-strong" model, in which only the "weak" beam is affected by the higher intensify "strong" beam [2], or soft Gaussian model [3], where one beam is assumed a priori to have a Gaussian shape. BeamBeam3D was the first parallel code that can be used to study this interaction fully self-consistently for both beams on high-performance computing platforms, including all the physical processes of long range off-centroid interactions, finite beam bunch length effects, and crossing angle collisions. The code has been used to study the beam-beam interactions at the world's highest energy hadron accelerator currently used for experiments, Fermilab's Tevatron, at SLAC's Positron-Electron Project (PEP-II), at Brookhaven National Laboratory's Relativistic Heavy Ion Collider, and at Japan High Energy Accelerator Research Organization (KEKB).

Studying the performance of high-end computing platforms has always been an active topic and a lot of studies have been conducted. Some of them focus on overall comparison of different architectures, such as vector platforms vs. scalar platforms [6,8,9,13]; some of them focus on the performance of a specific application [10,11]. This study focuses on BeamBeam3D, a parallel particle-in-cell code to simulate beam collision for high energy ring colliders. Simulating the beam-beam collision for millions of turns is extremely time-consuming and may take weeks or even months to finish. Therefore, it is critical to understand which architectures suit this code well and how to effectively utilize the available computing resources since high performance computing users today can choose between varieties of considerable different computing platforms to execute their codes. We select four leading computing architectures for this study, a custom-built massively parallel system, a

commodity cluster, an advanced parallel vector platform, and a BlueGene/L (BG/L). We're interested in the following questions, for which the results will be summarized in Section 4: i) How to partition the workloads among processors to achieve optimal performance? ii) Are the performance bottlenecks on these diverse architectures similar or not and how can they be addressed to improve performance? iii) How does the performance of these systems compare? and iv) What are the main architectural requirements of this challenging application? Sine the particle-in-cell approach used by BeamBeam3D is a common approach adopted by many scientific codes, the result will not only benefit BeamBeam3D, but also other particle-based codes in molecular dynamics, plasma physics and cosmology studies. Furthermore, it also helps computer designers to understand the application requirements to improve the next generation architecture design.

The rest of the paper is organized as follows: The computational method and parallel implementation are described in Section 2. In Section 3, the four platforms are introduced, followed by the discussion of workload partition among the processors, analysis of execution time, performance optimization, and performance comparison. Finally, in Section 4, we summarize our results.

2. Computational methods and implementations for BeamBeam3D

BeamBeam3D models the colliding process of two counter-rotating charged particle beams moving at speeds close to the speed of light. Under the paraxial approximation, for the relativistic charged beam, the electric forces and the magnetic forces will cancel each other within the beam. However, for the colliding beams, moving in the opposite directions, the electric forces and the magnetic forces add up. The resulting beam-beam force is a strongly nonlinear interaction that can significantly affect the motion of the charged particles. We use a multiple slice model to calculate the electromagnetic forces. In this model, each beam is divided into a number of slices along the longitudinal (Z direction in Fig. 2) direction in the moving frame. Each slice contains nearly the same number of particles at different longitudinal locations z. The colliding process for two beams that have been divided into 2 slices is illustrated in Fig. 1. During each step, only the red (gray) slices from opposite beams collide with each other.

There are two important domains in BeamBeam3D, particle domain and field domain. The particle domain is the configuration space containing the charged particles, and the field domain is the space where the electric field is generated by the charged particles. In the field domain decomposition, the whole computational domain is divided into a number of subdomains, and each subdomain together with the particles inside it is assigned to a processor. Since all particles are local to a processor, the Poisson equation, which has been used to compute the electric and magnetic forces for the field, is solved on the grid and the particles are advanced using the electromagnetic fields. However, in the next turn, the particles

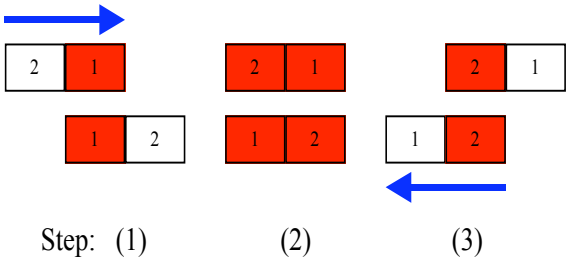

Figure 1. The colliding process for two 2-slice oppositely moving strong beams. It takes three steps to finish. The red (gray) slices are colliding slices. The blank slices are non-colliding slices. Beam1 is on top and Beam2 is at bottom.

belonging to a subdomain may have moved to other subdomains. Therefore, the processors of different subdomains have to exchange particles with each other. Due to the fact that in the accelerator, the lattice map outside the interaction point may cause significant particle movement, the effective communication pattern can end up as all-to-all communication and the data volume could be very large.

In the case of particle decomposition, only the particles are evenly distributed among the processors regardless of their physical positions; and the computational domain is shared by all processors, i.e., every processor owns the whole computational domain. To solve the Poisson equation, the particles are deposited onto the global computational grid, collected and broadcast to all processors. Each processor now owns the charge density distribution of the whole domain, and the Poisson equation is solved locally. This approach ensures perfect load balance but it does not take advantage of the parallelism in the solution of the Poisson equation.

BeamBeam3D adopts a novel particle-field decomposition approach to combine the advantages of both domain decomposition and particle decomposition, which has been demonstrated to deliver

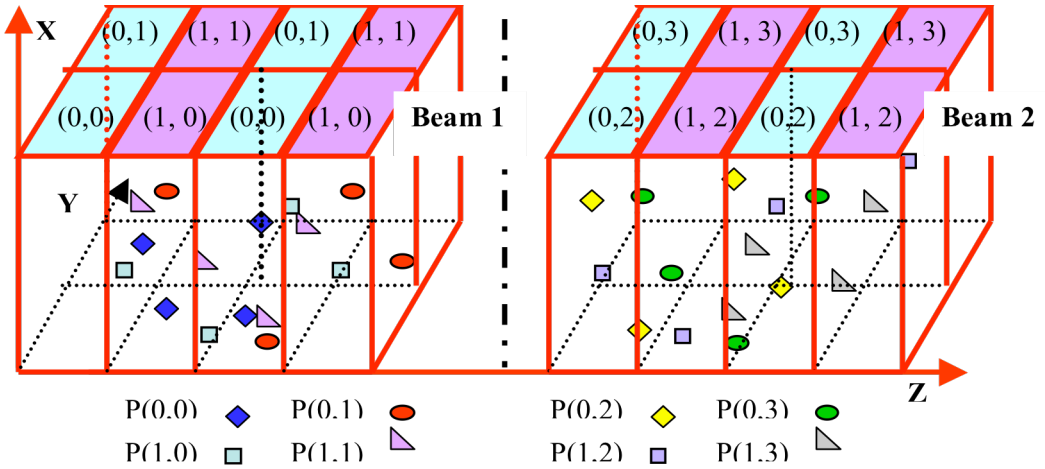

Figure 2. Schematics of the particle-field decomposition for 8 processors

better performance than either particle decomposition or domain decomposition alone [1,13]. In this approach, each processor possesses the same number of particles and the same number of computational grid points, i.e., a spatial subdomain of the same size. Fig. 2 shows a schematic plot of the particle-field decomposition among eight processors. The total number of processors is divided into two groups, with each group responsible for one beam. We furthermore divide each beam longitudinally (Z direction) into a specified number of slices (*Nslice* = 4 in Fig. 2). The processors in each group are arranged logically into a two-dimensional array P_z*P_y to partition the computational domain, with each column (P_y) of the array containing a number of slices which are assigned to this column of processors cyclically along the longitudinal direction. This gives a good load balance of slices among different column processors. Within each column, the computational grid associated with each slice is decomposed uniformly among all the column processors. This allows us to parallelize the solution of the Poisson equation.

The spatial coordinates of the particles for each processor may not fall within the spatial mesh domain of that processor. During the solution of the Poisson equation, the particles are first deposited onto the computational grid to obtain the charge density distribution. For the particles with spatial positions outside the local subdomain, an auxiliary computational grid is used to store the charge density. After the deposition, the charge density stored on the auxiliary grid will be sent to the processor owing that subdomain. With the charge density local to each processor, the Poisson equation is solved in parallel on a local subdomain using the shifted Green function method [1]. Since each processor contains the same number of computational grid points, the workload is well balanced among all processors. However, as in all domain decomposition approaches, load imbalance could occur across row processors when the number of colliding slices during the colliding process is not a multiple of the number of processor rows.

The solution of the electric potential on the local subdomain is sent to all processors and the electric field is calculated on the grid and interpolated onto individual particles of the opposite beam. The particles are advanced using the electromagnetic field and the external maps. Since each processor contains the same number of particles, the work of this process is also well balanced across processors. The volume of communication in the particle-field decomposition approach is proportional to the number of computational grid points instead of the number of moving particles in the domain decomposition approach. Since, in the study of beam-beam interactions, the number of particles is potentially much larger than the number of computational grid points, the particle-field decomposition approach can significantly reduce the communication cost in the simulation. The code is developed in F90 and MPI.

3. Performance analysis and discussion

In our tests, we only simulated 100 turns for the performance study since the time taken by each turn does not change much. The number of particles for each beam is five million and each beam is divided into eight slices longitudinally. Using more particles and longitudinal slices will reduce the random

sampling effects and improve the resolution in the simulation, but may also significantly increases the computing time. For most applications, a few million particles with five to ten slices will be sufficient for the convergence of the luminosity calculation. The computational grid size is 256*256*32 in X, Y, and Z direction.

3.1. Test platforms

We select four high-performance computing platforms, representing four different types of leading architectures.

- *Seaborg* is a massively parallel system located at Lawrence Berkeley National Laboratory (LBNL). It contains 380 SMP nodes which are interconnected with the IBM Colony-II switch using an omega-type topology. Each Nighthawk II node consists of sixteen 375MHz Power3-II processors. The peak performance of a processor is 1.5GFlops/s.
- *Jacquard* is a typical commodity-component based cluster platform also located at LBNL. Each node contains two 2.2 GHz Opteron processors connected via InfiniBand fabric in a fat-tree configuration. The peak performance of a processor is 4.4GFlops/s.
- *SX8* is an advanced vector platform located at High Performance Computer Center (HLRC) in Stuttgart, Germany. It contains 72 SMP nodes (8 processors) connected via a custom single-stage crossbar. The SX-8 processor operates at 2 GHz with peak performance of 16Gflops/s.
- *BlueGene/L* is novel computer architecture, located at Argonne National Laboratory. It uses the computationally less powerful but much more power efficient processor PowerPC440 running at 700 MHz with peak performance of 2.8GFlops/s. There are a total of 1024 dual PowerPC440 nodes connected by five special interconnect networks for I/O, debug, and interprocessor communication. A three dimensional torus is the major data communication network for applications. A collective network allows data to be sent from any node to all other nodes or a subset of nodes.

3.2. Partitioning the workloads among processors

One critical step to enable the applications to run well on parallel platforms is to be able to efficiently partition the workloads among the processors. In BeamBeam3D, the particles are evenly partitioned among the processors. However, in order to obtain optimal performance, we find that the partition of the field domain, i.e., the selected values of P_z and P_y, should be dynamically adjusted based on the number of processors actually used and consider both the computation and communication cost. This is different from purely pursuing workload balance. Fig. 3 shows the total running time on Jacquard for different combinations of P_z *P_y (P_z =1,2,4,8, P_z <= Nslice). We can clearly find that the performance is highly affected by the selection of P_z and P_y.

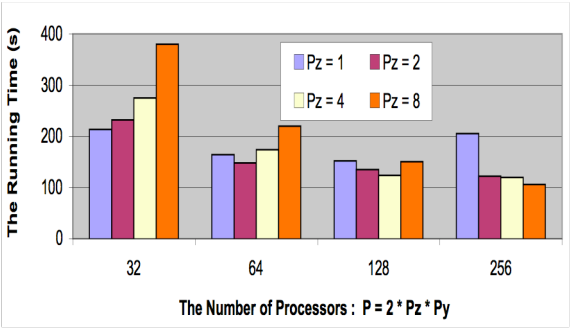

Figure 3. The performance on Jacquard for different partitions under different number of processors

As we know, the spatial coordinates of the particles belonging to a processor are not confined within its assigned computational domain and are scattered all over the computational space. Therefore, during the process of depositing the particles to the computational domain to obtain the charge density distribution, a processor responsible for a specified subdomain has to collect this information from all other processors (*deposit*). This process is organized into two phases, collecting in column direction first and then in row direction. After solving the Poisson equation locally with the collected charge density, the solution of the electric potential on the local subdomain is broadcast to the opposite group to compute the electric field and advance the particles (*scatter*). The number of messages and the message sizes in the *deposit* and *scatter* phases are closely related with the values of P_z and P_y.

For 32 processors, P_z = 1 delivers the best performance. This is because computation dominates the performance at this scale and most of the time is spent on solving the Poisson equation using the shifted Green function. In this case, reducing the computation time is most important. P_z = 1 means a slice will be partitioned among sixteen processors (in Y direction) and each one is responsible to solve the Poisson equation for 1/16 of the slice; while P_z = 8 indicates only two processors are available for a slice. Therefore, the time to solve the equation will be substantially increased.

However, with an increasing number of processors, a value of P_z >1 starts to improve the performance. For 64, 128, and 256 processors, selecting value of 2, 4, and 8 respectively will deliver the best performance. This is because communication gradually becomes the dominant performance factor. Assigning more processors in Z direction will lead to fewer messages and larger message sizes, which will reduce the communication cost. Table 1 shows the approximate numbers of messages per processor in different stages and corresponding message sizes. The *transpose* happens during the solution of the Greens function and Poisson equation. We can clearly find that the message sizes when $P_z = 8$ is at least eight times larger than the message sizes when $P_z = 1$. This is also true on other platforms. With the increase of the number of processors, assigning more processors in Z direction will help to improve the performance. For the remainder of this study we will use the performances obtained by the best combination of P_z and P_y.

Table 1. The No. of messages and message sizes (KB)

	P = 32				P = 256			
	$P_z = 1$		$P_z = 8$		$P_z = 1$		$P_z = 8$	
	Size	No.	Size	No.	Size	No.	Size	No.
Deposit	32	960	256	169	4	8192	32	1065
Scatter	32	1024	256	233	4	8192	32	1129
Trans-pose	8	450	512	30	1/8	3810	8	3600
Trans-pose	4	3840	256	256	1/16	32512	4	3840

3.3. Performance Bottleneck Analysis

In order to analyze the performance bottleneck, each turn of the beam collision is logically divided into the following major phases:

- *Preprocessing*: compute shift and sigma, transfer coordinates for particles, and update the physical domain size
- *Slicing*: divide the particles into *Nslice* slices
- *Greenf*: compute the Green function for the solver
- *DepositLocal*: deposit local particles into the field to obtain the charge density
- *DepositGlobal*: collect charge density from other processors for local subdomain
- *FieldSolver*: solve the Poisson equation
- *ScatterGlobal*: scatter the local electric potential to processors in opposite group
- *ScatterLocal*: apply the electric field to local particles
- *Postprocessing*: transfer the coordinates back, apply linear and non-linear map, apply radiation damping and quantum excitation
- *I/O*: output information to files

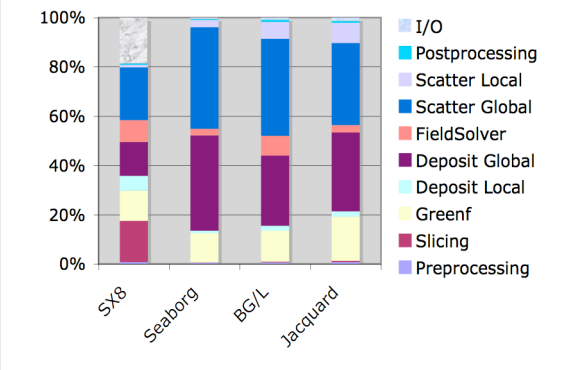

Figure 4. The pre-optimized time breakdowns for a 256-processor run

Fig. 4 shows the time distribution for a 256-processor run for these phases on four platforms. The time spent for each phase is the average time across all 256 processors. The breakdown for the SX8 differs substantially from the other platforms. The I/O phase, which takes little time on other platforms, consumes approximately 18% of the total cycles. The *slicing* phase and the *FieldSolver* phase on the SX8 also consumes higher percentage of the total cycles. In contrast to this, the *ScatterGlobal* and *DepositGlobal* communication phases perform much better on the SX8. The time breakdowns for Seaborg, BG/L, and Jacquard are much more similar. They are all dominated by communication phases. However, the percentages of time needed by the different phases still show differences. For example, BG/L spent much high percentage of time on *FieldSolver* phase while Jacquard consumes higher percentage on *Greenf*. Note that the higher percentage does not necessarily mean that their actual running time is higher.

The different time breakdowns in Fig. 4 indicate that each architecture may exhibit its own unique bottlenecks and need to be solved individually though all platforms suffer from the high communication cost. Now let's examine how to improve the performance on the SX8. Different steps in the optimization process on the SX8 are shown in Fig. 5. The first version labeled as *Vectorized.*which runs three times faster than the original version, is obtained by rewriting many of the loop bodies using coding techniques such as dividing large loop bodies into several smaller loops, extracting complex math operations out of loop bodies, and adding compiler directives. Its breakdown shows that

the *slicing* phase, which cannot be easily vectorized, takes almost 17% of the total running time. In order to vectorize this phase, the virtual processor concept is used [4]. Each element of the vector register is viewed as a virtual processor. Each virtual processor is then assigned a portion of the particles and a set of independent auxiliary data structures so that it can work exactly as a processor. This optimization reduces the time of the slicing phase from over 17% to less than 1% of the total running time. The next step is to optimize the I/O phase. At the end of each turn, the processor of rank zero collects information from all other processors and writes them to multiple files. An easy optimization is to aggregate the messages to reduce output frequency. This approach turns out to be very effective.

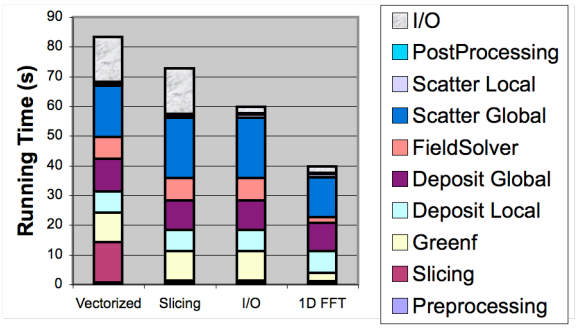

Figure 5. The process of optimizing performance on the SX8

Table 2. The 1D FFT time and total running time for 256 processors

	1D FFT Time (s)			
	SX8	Seaborg	BlueGene/L	Jacquard
Original	13.2	15.4	22.6	3.9
Optimized	0.9	3.3	3.9	1.5
Speedup	14.6	4.7	5.8	2.6
	BeamBeam3D Running Time (s)			
Original	60.0	573.0	389.0	116.6
Optimized	40.0	495.0	362.0	116.1
Speedup	1.50	1.16	1.07	1.00

Finally, we optimize the 1D FFTs which are heavily used in the *FieldSolver* and *Greenf* phases. The original implementation could not be vectorized and becomes a performance bottleneck. We replace them with vendor supplied 1D FFT routines. From the *1D FFT* time breakdown in Fig. 5, we see that the time for *FieldSolver* and *Greenf* phases has been significantly reduced. In addition, the time for *ScatterGlobal* phase is also reduced. This is related to the load imbalance in the *FieldSolver* phase. Since only processors that own colliding slices will participate in solving the Poisson equations while other processors simply wait to receive the results, some load imbalance will occur and this is difficult to be avoided in multi-slice beam models. Therefore, the times needed for the *FieldSolver* phase are different across processors, but the sum of *FieldSolver* and *ScatterGlobal* phases is almost equal. If the process of solving the Poisson equation can be accelerated, the waiting time in the *ScatterGlobal* phase will also be correspondingly reduced.

The optimizations for the *I/O* and *Slicing* phases on SX8 have almost no effect on Seaborg, BG/L, and Jacquard since both phases are insignificant there. However, the use of vendor supplied FFT benefits all platforms. Table 2 displays the 1D FFT running time and the total application running time before and after using vendor supplied FFT (original vs. optimized). The effect of this optimization varies widely across different platforms. In the best case, the 1D FFT time has been improved 14.6 times, leading to 33% reduction in total running time on SX8. In the worst case, the 1D FFT time has been reduced 2.6 times but only has negligible effect on total running time on Jacquard. The 1D FFT is not a problem there, the communication is. We have tried to use fewer larger messages to reduce communication cost through optimal processor partitions. Another way is to take advantage of the communication hierarchy to allow as many as possible communications occur inside SMP nodes instead of between SMP nodes. This work is still under progress.

3.4. Performance comparison and architecture implication

Fig. 6 displays the best performance obtained on each platform for different number of processors in log-log scale. Seaborg performs worst among all four platforms. BG/L is 30%-40% better than Seaborg and Jacquard is around 65% better than BG/L. The SX8 delivers the best performance and is about two times faster than Jacquard and more than ten times better than Seaborg does. The crossbar interconnect on the SX8 provides the highest bisection bandwidth among these four platforms, 4096GB/s for 256 processors; while Seaborg provides the worst bisection bandwidth 32GB/s for 256 processors. Correspondingly, the communication times needed for a 256 processor run are 30.1, 60.9, 199.7, 378.3 seconds on SX8, Jacquard, BG/L, and Seaborg, respectively. The results indicate that higher bisection bandwidth is essential for this kind of application to work well.

The parallel efficiency degrades very fast for all systems. If we use the performance of 32 processors as the reference point, the parallel efficiency drops to around 22% for 256 processors. The poor scaling behavior is mainly attributed to the non-decreasing communication time, which consumes only around 20% of the total cycles for 32 processors but increased to over 60% for 256 processors.

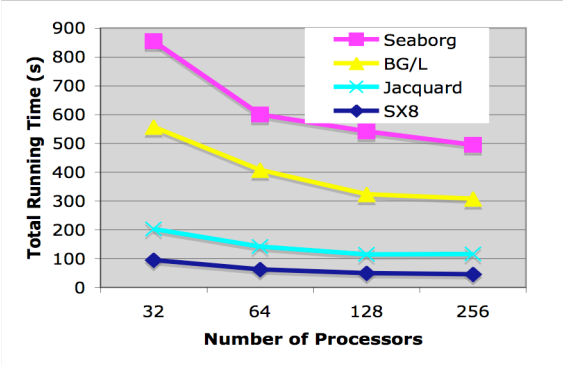

Figure 6. The achieved best performance on different platforms

The communication is dominated by global gather/scatter operations. Once the field domain size has been defined, the total communication volume is almost fixed regardless of the concurrency. Table 3 shows the total communication volume sent by the processor on the critical path in the *DepositGlobal, ScatterGlobal, FieldSolver*, and *Greenf* phases on Jacquard.

Table 3. The measured and predicted communication performance on Jacquard

	32	64	128	256
P_z	1	2	4	8
Vol. (MB)	75.33	74.03	74.17	75.81
Time(s) Measured	54.01	49.86	52.52	53.73
Time(s) Modeled	47.21	45.51	45.16	45.97
Time(s) Modeled Bandwidth * 2	26.08	24.75	24.35	24.71
Time(s) Modeled Latency * 0.5	44.73	43.52	43.38	44.24

The total amount of data sent by a processor is around 75MB. The time needed to send the messages is almost flat. Remember that we use the optimal partition for each case. If not, the communication time could go much higher. We develop a simple model to test how the change of the bandwidth and latency will affect the communication time. The model is based on the measured latency and peak bandwidth obtained through a MPI performance probe, in which every processor in a SMP node will pingpong messages with its pair in another SMP node so that contention effect will be included. This simple model was able to correctly predict the flat timing increase trend with error bound of 17% on Jacquard. This model also works well on other platforms. Based on this model, if we double the available bandwidth, the communication time will reduce over 50%. This indicates that this demanding code should benefit substantially from the higher bandwidth, at least at the scale we tested.

Since the communication volume does not change much with the number of processors, using more processors will decrease the message size and increase the number of messages. Therefore, the performance will become more latency sensitive. In table 3, we show the predicted communication time by halving the network latency on Jacquard. We find that at this scale the latency only has a slight effect to the total communication time, which is reduced less than 5%. However, the latency effect will become explicit when we use large number of processors. Table 4 shows the latency effect for the 2048-processor case. If we reduce the latency to its half, the communication time could become 25 - 81% lower, depending on the partition used. Here, the latency refers not only the network latency but also the software overhead to send /receive the messages. One possible approach to reduce latency is to move the global scatter/gather operation into the network and exploit the concurrency provided by some advanced interconnects [14,15].

Table 4. The predicted latency effect on 2048 processors on Jacquard

P_z	1	2	4	8
Time (s), Latency * 1.0	371	170	95	65
Time (s), Latency * 0.5	204	104	67	52
Ratio	1.82	1.63	1.42	1.25

Regarding local operations, the times needed for a 256-processor run is 14.9, 46, 108, and 116 seconds on SX8, Jacquard, BG/L, and Seaborg, respectively. The vector architecture of SX8 really excels itself for this code after spending significant effort on optimizing the code. Providing the *virtual vector units* [16] inside the superscalar architecture may be one promising approach for future processor designs. The achieved percentage of peak on this four platforms ranges from 6.71% to 11.61%. This relatively low percentage of peak performance is partly due to the large amount of

data movement in some phases, which consumes a lot of cycles but do not contribute to Mflops at all.

4. Summaries

In this paper, we analyzed the performance characteristics of BeamBeam3D on four leading computing platforms. First, the strategy to partition the workload among the processors has been examined. We find that, in order to achieve optimal performance, purely pursuing workload balance is not enough. Instead, we should focus on alleviate the performance dominant factor, enabling the partition to minimize the communication time or computation time whichever becomes dominant. Therefore, the optimal partition is dynamically changed with the number of processors used and the platforms to work on.

We also find that the code may exhibit different performance bottlenecks across architectures. Some negligible phases on one platform may cause serious performance problems on another one and have to be addressed individually. The optimizations applied on one platform may or may not be effective on other platforms. For example, using the vendor-supplied FFT improves the total running time 33% on SX8, but has almost no effect on Jacquard.

The best performance is delivered on SX8. For a 256-processor run, each turn takes about 0.45 seconds, which is about 2.6 times better than Jacquard, 6.8 times better than BG/L, and 11 times better than Seaborg. But the parallel efficiency degrades fast with the increase of the number of processors. The communication becomes a scaling bottleneck. Results indicate that providing higher bisection bandwidth should significantly improve the application performance, such as SX8 does. However, with large number of processors, the code's performance will become more sensitive to network latency. Therefore, in order to run this demanding code well on large number of processors, the platform should provide much lower latency than the current machines provide. Moving the global scatter/gather operation into the network and fully take advantage of the network concurrency may be able to help to lower the latency. SX8 also excels on the local computations. Its vector processing units perform significantly better for this application than the superscalar processors. Providing the *virtual vector units* inside the superscalar architecture may be one approach for future processor designs.

ACKNOWLEDGMENT

We would like to thank NERSC, Argonne National Laboratory, and HLRS to provide us access to their platforms. We would also like to thank anonymous reviewers for their insightful comments and suggestions.

REFERENCES

[1] J. Qiang, M.A. Furman, R.D.Ryne, "A parallel particle-in-cell model for beam-beam interaction in high energy ring colliders", J. Comput. Phys. 198 (2004) 278-294.

[2] K. Hirata, H. Moshammer, F. Ruggiero, "A symplectic beam-beam interaction with energy change", Particle Accel. 40 (1993) 205-228.

[3] M.A. Furman, "Beam-beam simulations with the gaussian code TRS", LBNL-42669, CBP Note272, 1999.

[4] H. Shan, E. Strohmaier "Performance Characterization of Cray X1 and Their Implications for Application Performance Tuning", International Conference of Supercomputing, Malo, France, June 2004.

[5] J. Berkowitz, "Model Colliders", DEIXIS 2004-2005 THE DOE Computational Science Graduate Fellowship ANNUAL.

[6] L. Oliker, J. Carter, M. Wehner, A. Canning, et. al., "Leading Computational Methods on Scalar and Vector HEC Platforms", SC 2005, Seattle, Washington, Nov. 2005.

[7] E. Strohmaier, J. J. Dongarra, H. W. Meuer, and H. D. Simon, "Recent Trends in the Marketplace of High Performance Computing", Parallel Computing, vol. 31, Mar. 2005.

[8] T. H. Dunigan, M. R. Fahey, J. B. White III, P. H. Worley, "Early Evaluation of the Cray X1", SC2003, Phoenix, AZ, Nov. 2003.

[9] K. Davis, A. Hoise, G. Johnson, D. J. Kerbyson, M. Lang, S. Pakin, F. Petrini, "A performance and scalability analysis of the BlueGene/L architecture", SC2004, Pittsburgh, PA.

[10] T. Pohl, F. Deserno, N. Thürey, U. Rüde, P. Lammers, G. Wellein, and T. Zeiser, "Performance evaluation of parallel large-scale lattice boltzmann applications on three supercomputing architectures", SC2003, Phoenix, AZ, Nov. 2003.

[11] M. R. Fahey, J. Candy, "GYRO: A 5-D gyrokinetic-maxwell solver", SC2003, Phoenix, AZ, Nov. 2003.

[12] A Science-Based Case for Large-Scale Simulation (SCALES). http://www.pnl.gov/scales.

[13] L. Oliker, A. Canning, J. Carter, J. Shalf, and S. Either, "Scientific computations on modern parallel vector systems", SC 2004, Pittsburgh, PA.

[14] V. Tipparaju, J. Nieplocha, "Optimizing All-to-All Collective Communication by Exploiting Concurrency in Modern Networks", SC05, Seattle, Washington, Nov. 2005

[15] A. Manidala, J. Liu, and D. K. Panda, "Efficient Barrier and Allreduce on IBA clusters using hardware multicast and adaptive algorithms", IEEE Cluster Computing, San Diego, CA, 2004.

[16] C. W. McCurdy, R. Stevens, H. Simon, W. Kramer, D. Bailey, W. Johnston, C. Catlett, R. Lusk, T. Morgan, J. Meza, M. Banda, J. Leighton, and J. Hules, "Creating Science-Driven Computer Architecture: a New Path to Scientific Leadership", http://www.nersc.gov/news/reports.